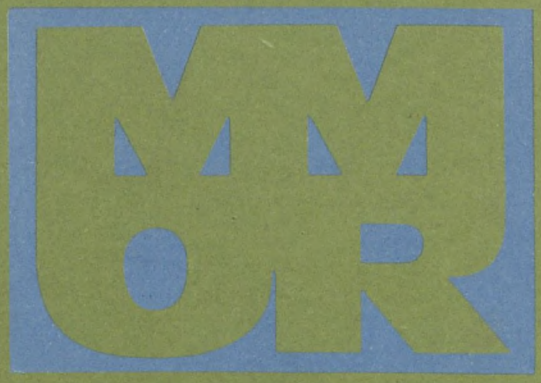
Mathematical Methods of Operations Research 2



LÁSZLÓ BÉLA KOVÁCS

COMBINATORIAL METHODS OF DISCRETE PROGRAMMING

Akadémiai Kiadó, Budapest

LÁSZLÓ BÉLA KOVÁCS

COMBINATORIAL METHODS OF DISCRETE PROGRAMMING

Discrete programming deals with optimization problems in which all or some of the variables take integer values.

Applicability: This field is a rapidly developing one because of its wide direct applicability and because of its close links with many other mathematical subjects.

Concise, comprehensive: This is a concise yet comprehensive textbook of the theory and practice of combinatorial methods of discrete programming with emphasis being placed on efficiency.

Coverage: After the presentation of many models one chapter is devoted to each of the basic methods, viz. implicit enumerations, branch and bound algorithms, dynamic programming. The following chapters consist of refinements such as Benders decomposition and a number of algebraic methods. Heuristic methods, specially structured problems, and complex algorithms are provided in further chapters. Finally, there is an annotated bibliography on the recent developments of discrete programming.

Descriptions, illustrative examples: All of the methods are described in theoretical frameworks but they can also be found in the form of step-by-step algorithms. Their illustration by means of numerical examples is an important asset of the volume.

AKADÉMIAI KIADÓ
BUDAPEST

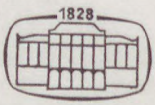
COMBINATORIAL METHODS OF DISCRETE PROGRAMMING

MATHEMATICAL METHODS OF OPERATIONS RESEARCH

Series Editor: A. PRÉKOPA

VOL. 1. STUDIES ON MATHEMATICAL PROGRAMMING
Edited by: A. PRÉKOPA

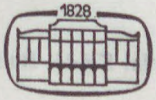
VOL. 2. COMBINATORIAL METHODS OF DISCRETE PROGRAMMING
LÁSZLÓ BÉLA KOVÁCS



AKADÉMIAI KIADÓ • BUDAPEST 1980

COMBINATORIAL METHODS
OF
DISCRETE PROGRAMMING

LÁSZLÓ BÉLA KOVÁCS



AKADÉMIAI KIADÓ • BUDAPEST 1980

ISBN 963 05 2004 4

Printed in Hungary

CONTENTS

PREFACE 11

Chapter 1

MODELS OF DISCRETE PROGRAMMING FOR ECONOMIC PROBLEMS AND FOR CERTAIN PROBLEMS IN MATHEMATICAL PROGRAMMING

1.1 General form of discrete programming problems 13
1.2 The knapsack problem 15
1.3 The cargo loading problem 18
1.4 The travelling salesman problem 19
1.5 Fixed-charge problems 22
1.6 An investment problem 24
1.7 A plant location problem 26
1.8 The empty container problem 27
1.9 Alternative constraints 29
1.10 Optimization of nonlinear separable functions 30
1.11 Conditional constraints 31
1.12 Minimization of concave functions on a convex polyhedron 31

Chapter 2

SINGLE BRANCH IMPLICIT ENUMERATION METHODS

2.1 Partial and lexicographic ordering of vectors 33
2.2 The Lawler–Bell algorithm 35
2.3 Summary of the Lawler–Bell algorithm and some remarks 38
2.4 The concept of the ordered pseudo-solution 41
2.5 Skeleton of enumeration algorithms 48
2.6 Application of enumeration algorithms to pure 0–1 problems 52

Chapter 3

BRANCH-AND-BOUND ALGORITHMS

3.1 Introduction 57
3.2 The branch-and-bound principle 57
3.3 Problem solving by the branch-and-bound principle 62
3.4 Solution of the knapsack problem 64
3.5 Solution of the travelling salesman problem 69
3.6 The method of Land and Doig for the solution of mixed integer problems 78
3.7 An improved branch-and-bound algorithm for mixed integer problems 86

Chapter 4

DYNAMIC PROGRAMMING AS A TOOL FOR SOLVING DISCRETE PROGRAMMING PROBLEMS

4.1 Introduction 96
4.2 Optimality principle of dynamic programming 96
4.3 Solution of the knapsack problem by dynamic programming 99
4.4 Solution of discrete problems of several constraints 102
4.5 Reducing the number of optimum functions 105
4.6 Extension of the single optimum function method to the bounded variable case 108
4.7 Further improvement of the single optimum function method 118
4.8 Application of the shortest path algorithm in dynamic optimization 121
4.9 Extension of group theoretic algorithms to general linear integer programming problems 129

Chapter 5

A MULTIPHASE DUAL ALGORITHM

5.1 Introduction 132
5.2 Surrogate constraints 132
5.3 A heuristic procedure for finding strong s-constraints 136
5.4 Another definition of s-constraint strength 138
5.5 Tests 142
5.6 Description of the algorithm 147

Chapter 6

A MODIFIED ADDITIVE ALGORITHM

6.1 Introduction 154
6.2 Notation, notions 154
6.3 Description of the algorithm 157
6.4 Justification of the algorithm 159
6.5 A numerical example 160

Chapter 7

BENDERS DECOMPOSITION

7.1 Introduction 163
7.2 Problem formulation 163
7.3 A reformulation of the problem 164
7.4 Preliminary lemmata 167
7.5 An algorithm for mixed integer problems 171
7.6 A numerical example 174

Chapter 8

MODIFIED FILTER METHOD

8.1 Problem formulation 182
8.2 Optimal indexing 185
8.3 Solution-tree and pseudo-solution tree 185
8.4 Use of the branch-and-bound principle 191
8.5 Tests 192
8.6 Description of the algorithm 196
8.7 A numerical example 200

Chapter 9

HEURISTICS IN DISCRETE PROGRAMMING

9.1 Introduction 204
9.2 Local search techniques 204
9.3 Relaxation methods 212
9.4 A tightening method 216
9.5 Probabilistic methods 218
9.6 Learning procedures 219
9.7 Testing and evaluating 220

Chapter 10

A NEW SOLUTION FOR THE GENERAL SET COVERING PROBLEM

10.1 Introduction 221
10.2 Problem and terminology 221
10.3 Applications 222
10.4 Survey of methods 224
10.5 A heuristic method 225
10.6 An exact solution for the general set covering problem 228

Chapter 11

COMPLEX ALGORITHMS

11.1 Introduction 230
11.2 Heuristic procedures for obtaining feasible solutions 230
11.3 *s*-Constraints 231
11.4 Tests 231
11.5 Free variable choice rules 231
11.6 Restricted enumeration 234
11.7 More flexible enumeration procedures 236
11.8 An adaptive approach to subproblems of small size 239
11.9 Mixing of algorithms 240
11.10 Man–machine interactions 241
11.11 Design of algorithms for discrete programming 242
11.12 Computer experience 243

REFERENCES TO CHAPTERS 1–11 245

Chapter 12

RECENT DIRECTIONS IN DISCRETE PROGRAMMING 259

12.1 Basic research supporting algorithms of discrete programming 259
12.1.1 Complexity and approximation of combinatorial algorithms 259
12.1.2 Probabilistic and asymptotic methods 260
12.1.3 Integer polyhedron 261
12.2 Advances in general discrete programming algorithms 261
12.2.1 Implicit enumeration 261
12.2.2 Branch-and-bound algorithms 262
12.2.3 Dynamic programming 263
12.2.4 Cutting planes 263
12.2.5 Heuristics and approximations 264

12.3 Development of algorithms for specially structured discrete problems 264
12.3.1 Networks and graphs 264
12.3.2 Knapsack problem 265
12.3.3 Scheduling 265
12.3.4 Travelling salesman and other routing problems 266
12.3.5 Set covering, partitioning and packing 267
12.3.6 Quadratic assignment problem 267
12.3.7 Further typical applications 268
12.4 Extensions, experimentation 268
12.4.1 Sensitivity and parametric analysis, duality 268
12.4.2 Nonlinear integer problems 269
12.4.3 Miscellaneous 270

REFERENCES TO CHAPTER 12 271

INDEX 281

PREFACE

Problems in discrete programming† relating to investment, production scheduling, line balancing, fixed cost, cutting stock, etc. arose shortly after the first successful applications of linear programming, i.e. in the early 50's. Many heuristic algorithms – including rounding procedures and some kind of cutting methods – were attempted with little success either in theory or in practice.

The first exact algorithm (a cutting-plane procedure) for solving discrete programming problems was written by Gomory in 1958. This was followed by the pioneering works of Land and Doig in 1960 providing the basis of the branch-and-bound algorithm, and that of Balas containing one of the first implicit enumeration methods. Since that time hundreds of papers have demonstrated the importance and the difficulty of the subject. Even though all linear programming problems are solved by different versions of the same basic idea, the simplex method, it is not so with the discrete problems. There are at least four large and many smaller groups of algorithms for solving pure and mixed integer programming problems, i.e. when all or part of the variables may take only integer values. None of these methods has turned out to be uniformly better than the others; in fact, it seems to be increasingly clear that only complex algorithms can give satisfactory results for a wide variety of problems. Some important and frequently occurring discrete problems, however, deserve the construction of special algorithms which are substantially more effective than the general ones.

Discrete programming is closely related to many other mathematical fields. All problems of network flows and (discrete) dynamic programming, furthermore all construction problems of combinatorics and graph theory can be formulated as discrete programming problems. On the other hand, integer programming takes its tools from all these fields besides using the methods developed specifically for discrete programming.

The present work is partly a textbook and partly a monograph:
– it is a textbook because the first chapter gives a detailed introduction to the models and problems of integer programming and the further three chapters discuss three main types of algorithms, viz. implicit enumeration, branch-and-bound algorithms, and dynamic programming algorithms;

† The terms discrete programming and integer programming will be used synonymously, see Chapter 1.

– it can also be considered a monograph because several new results of the author, further development of existing algorithms, new procedures, frameworks of further research, heuristic procedures, complex algorithms, etc. are published here for the first time.

As a result of its twofold character the book is likely to be of interest to graduate and postgraduate students, and to mathematicians, engineers and economists working in the field of operations research.

For mathematical background, linear programming though not indispensable, is advisable; an amount of knowledge in group theory is necessary for the second part of Chapter 4. The actually presupposed knowledge is very little – mainly in combinatorics and n-dimensional Euclidian spaces. A certain capability for mathematical abstraction and model building is needed.

The contents of the book are as follows. Chapter 1 describes the general form of discrete programming problems together with some basic problems including models for dealing with practical problems. A general framework for algorithms of implicit enumeration is given in Chapter 2. Chapter 3 contains the branch-and-bound principle and its application to pure and mixed integer problems. The main ideas are also worked out for some special problems. Chapter 4 describes the application of dynamic programming in the original and extended form using network flows and group theory. The fifth chapter is devoted to an algorithm of F. Glover, which is the only practically unchanged presentation in the book. Different ways of constructing consequences – so-called s-constraints – of a given inequality system are also included. Chapters 6 and 8 are used for describing two algorithms of Balas both of which are substantially revised. Chapter 7 contains a detailed presentation of the Benders decomposition for mixed integer problems. The main directions and the role of heuristics in discrete programming are outlined in Chapter 9; this chapter discusses heuristic tools of exact algorithms; further practical problems are to be found here. Chapter 10 is devoted to a new solution of the set-covering problem giving a good example for the symbiosis of fast heuristic with exact methods. Chapter 11 summarizes the experience of the author in designing complex algorithms. Finally, Chapter 12 is an account of new directions and results in discrete programming with extensive references to the latest literature.

Since the character of the book is mainly combinatorial, cutting plane algorithms are not covered in detail but an outline and several references may be found in Chapter 5. All algorithms are illustrated by worked examples to facilitate understanding. The author regularly holds course at the Eötvös Loránd University, Budapest, for senior mathematics students specializing in operations research. About half the material of this book is covered in the courses.

A Hungarian version *(A diszkrét programozás kombinatorikus módszerei)* of the present book has appeared as lecture notes in the series "Mathematical Methods of Operations Research" (János Bolyai Mathematical Society, Budapest, 1969). This earlier version has been completely revised, enlarged, and the last four chapters newly added.

László Béla Kovács

CHAPTER 1

MODELS OF DISCRETE PROGRAMMING FOR ECONOMIC PROBLEMS AND FOR CERTAIN PROBLEMS IN MATHEMATICAL PROGRAMMING

The present chapter contains the general form of discrete programming and presents models which demonstrate the importance of the field.

Throughout the book matrices and vectors will be denoted by bold face capital and bold face lower case letters respectively, **A**, **B**, **C**, and **a**, **x**, **y**, etc. The vectors are mainly represented by column vectors. For transposition the letter T is used.

1.1 General form of discrete programming problems

The problems of mathematical programming are those of minimization or maximization of a function of several variables, on a domain determined mostly by equations and inequalities, containing similar functions of several variables. There may be some additional requirements which are expressed in logical form.

A mathematical programming problem that contains discrete variables, i.e. variables taking only a finite or countably infinite (e.g. 0, 1, 2, . . .) number of values, is called a discrete programming problem.

The problems of discrete programming can be summarized in the following form:

Minimize

$$f(\mathbf{x}, \mathbf{y}) \tag{1.1}$$

subject to

$$g_i(\mathbf{x}, \mathbf{y}) \geqq 0 \quad (i = 1, 2, \ldots, m) \tag{1.2}$$

$$x \in S \tag{1.3}$$

where

$$\mathbf{x} = (x_1, x_2, \ldots, x_p)$$

$$\mathbf{y} = (y_1, x_2, \ldots, y_q)$$

$p > 0$, $q \geqq 0$. S is a finite set of p-dimensional vectors, f and g_i are given functions of $p + q = n$ variables. In most cases the number of elements of S is finite or it can be reduced to the finite case. The set S may be very simple, for example, the set of p-dimensional 0–1 vectors, i.e. it consists of all p-dimensional vectors the components of which are equal to 0 or 1. In other cases the definition of set S may contain complicated logical expressions, the algebraic formulation of which is not reasonable.

If the number of continuous variables $q > 0$, then the problem is called a mixed variable problem. In the opposite case, i.e. if there are only discrete variables

$x_1, \ldots, x_p$ in the problem, then it is called a pure integer (discrete) programming problem.

The term "integer programming" is also used as a synonym of discrete programming. The reason is that from any problem in which the variables may take on a finite number of given values, another problem can be deduced in which the variables may take on only integer values. Let us suppose, for example, that variable x may take on the values $a_1, a_2, \ldots, a_k$ then let

$$\begin{aligned} x &= a_1u_1 + a_2u_2 + \ldots + a_ku_k \\ u_1 &+ u_2 + \ldots + u_k = 1 \\ u_j &\in \{0, 1\} \quad (j = 1, 2, \ldots, k) \end{aligned} \tag{1.4}$$

that is, the discrete variable x may be substituted by several 0–1 variables. Similarly if

$$x \in \{0, 1, 2, \ldots, k\},$$

then

$$\begin{aligned} x &= u_1 + u_2 + \ldots + u_k \\ u_j &\in \{0, 1\} \quad (j = 1, 2, \ldots, k) \end{aligned}$$

that is, any problem containing arbitrary bounded integer variables can be reduced to a problem containing only 0–1 variables. But there is a more economical way of reduction when a substantially smaller number of 0–1 variables are needed. Namely, let r be a positive integer such that the inequalities

$$2^{r-1} \leqq k < 2^r$$

are satisfied. Then

$$\begin{aligned} x &= 2^{r-1}u_1 + 2^{r-2}u_2 + \ldots + 2^1u_{r-1} + 2^0u_r \\ u_j &\in \{0, 1\} \quad (j = 1, 2, \ldots, r). \end{aligned} \tag{1.5}$$

This is the so-called binary representation. This is unique in the sense that there is a one-to-one correspondence between the integer values of the interval $[0, 2^{r-1}]$ and the r dimensional vectors of 0–1 components. In other words there is no better representation with 0–1 variables than that of (1.5). The number of 0–1 variables needed for the substitution of the integer variable x ($0 \leqq x \leqq k$) is therefore

$$r = [\log_2 k] + 1$$

where $[a]$ represents the integer part of a, that is, the largest integer less than or equal to a.

This is one reason why we shall consider mostly the problems of 0–1 variables. The other reason is that the solution of the continuous version of the problem is the least meaningful in this case. In some cases however it is worth considering the non 0–1 case separately.

The rest of the chapter will be devoted to simplified models and applications, through which some questions and difficulties of discrete programming will be illustrated.

1.2 The knapsack problem

In the literature the one-constraint, linear pure 0–1 discrete programming problem is called the knapsack problem; it is so named because it was formulated first as a problem of hikers. By analysing the problem we can show many interesting points and several difficulties of discrete programming. On the other hand there are some methods for the problems having several constraints in which the one-constraint problem should be solved several times. This will therefore be considered as a test problem and several exact methods will be illustrated by its use. In the present paragraph a simple approximate solution will be obtained. This gives an estimation for the objective function, which will be useful even for the exact methods.

The verbal description of the problem is as follows. A hiker wants to choose his equipment from n different objects in order to carry a maximum use-value in his sack without overloading himself.

Notation

n Number of objects
a_j Weight of object j
c_j Use-value of object j
K Maximal permitted weight of the sack content

$$x_j = \begin{cases} 1 & \text{if the hiker takes object } j \\ 0 & \text{otherwise.} \end{cases}$$

With the aim of obtaining the maximal use-value without exceeding the weight limit

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

$$\sum_{j=1}^{n} a_j x_j \leqq K \tag{1.6}$$

$$x_j \in \{0, 1\} \quad (j = 1, 2, \ldots, n) .$$

The problem could have been formulated in a different way. We cannot get an item of equipment of maximal use-value and of minimal weight, because there may be only one objective function in one problem. But we may wish to determine a set of objects having at least a given use-value and of minimal weight. Another way of formulating the problem is to construct a new objective function which is a linear combination of the value function and the weight function (with negative sign). In this case the maximal capacity as a requirement can be taken into account. In both cases we shall obtain a problem similar to that of (1.6).

Let us return to the mathematical problem (1.6). If we forget the original meaning of the coefficients c_j and a_j, they may be negative. It can easily be seen, however,

that (1.6) may be reduced to the case of positive coefficients. First of all, if for variable x_j the coefficients $c_j \leqq 0$ and $a_j \leqq 0$, then by introducing a new variable

$$x_j' = 1 - x_j$$

the coefficients will become nonnegative and the variable x_j' may also take on the values 0 and 1. (Naturally, the right hand side K must be changed.) Furthermore, if $c_j > 0$ and $a_j \leqq 0$, then in the optimal solution x_j must be 1. Similarly if $c_j \leqq 0$ and $a_j \geq 0$, then there is at least one optimal solution of the problem in which the corresponding x_j takes on the value 0. We may therefore suppose, without restricting the generality, that

$$c_j > 0, \quad a_j > 0 \quad (j = 1, 2, \ldots, n).$$

Furthermore, let us suppose that the order of the objects is already such that

$$\frac{c_1}{a_1} \geqq \frac{c_2}{a_2} \geqq \ldots \geqq \frac{c_n}{a_n}. \tag{1.7}$$

In a simple case we can directly determine the optimal solution of (1.6).

Lemma 1.1: *If there exists an integer* r, $0 < r \leqq n$ *such that*

$$\sum_{j=1}^{r} a_j = K \tag{1.8}$$

and the ordering (1.7) *holds, then*

$$x_1 = x_2 = \ldots = x_r = 1$$

$$x_{r+1} = x_{r+2} = \ldots = x_n = 0$$

is an optimal solution of problem (1.6).

Proof: Let us consider any feasible solution of (1.6)

$$x_{j_1} = x_{j_2} \ldots = x_{j_s} = 1$$

$$x_j = 0 \quad (j \neq j_1, j_2, \ldots, j_s).$$

We have to show that

$$\sum_{p=1}^{r} c_p \geqq \sum_{q=1}^{s} c_{j_q}. \tag{1.9}$$

Let

$$J_1 = \{j \mid j \leqq r\}, \; J_2 = \{i_1, j_2, \ldots, j_s\}, \qquad J = J_1 \cap J_2$$

and

$$K' = \sum_{p \in J} a_p.$$

Then

$$\sum_{j \in J_1 - J} c_j = \sum_{j \in J_1 - J} \frac{c_j}{a_j} a_j \geqq \frac{c_r}{a_r}(K - K') \geqq \frac{c_r}{a_r} \sum_{j \in J_2 - J} a_j = \sum_{j \in J_2 - J} \frac{c_j}{a_j} a_j = \sum_{j \in J_2 - J} c_j$$

If $\sum_{j \in J} c_j$

is added to both ends of this series of inequalities, then the desired result (1.9) is obtained.

If the supposition of Lemma 1.1 is not satisfied, then the structure of optimal solutions may be different, i.e. a variable may take on the value 0 while another one with larger subscript takes on the value 1. (Variables are indexed according to the ordering (1.7)). For example, in the problem

$$\max 12x_1 + 9x_2 + 8x_3$$

subject to

$$6x_1 + 5x_2 + 5x_3 \leqq 10$$

$$x_j \in \{0, 1\} \quad (j = 1, 2, 3)$$

the ordering holds

$$\frac{12}{6} > \frac{9}{5} > \frac{8}{5}$$

and the solution $x_1 = 1, x_2 = x_3 = 0$ gives only an objective function value 12, but the solution $x_1 = 0, x_2 = x_3 = 1$ gives a higher result 17.

Now we give an estimation of the optimum of (1.6) for the case when condition (1.8) does not hold. This will be helpful later in seeking the optimal solution.

Let us denote the set of feasible solutions of (1.6) by R, that is,

$$R = \left\{ \mathbf{x} \,\middle|\, \sum_{j=1}^{n} a_j x_j \leqq K, \quad x_j \in \{0, 1\} \quad (j = 1, 2, \ldots, n) \right\}.$$

Lemma 1.2: *If r is an integer such that $0 \leqq r \leqq n$ and*

$$\sum_{j=1}^{r} a_p < K < \sum_{j=1}^{r+1} a_p,$$

then

$$\max_{x \in R} \mathbf{c}^T \mathbf{x} \leqq \sum_{j=1}^{r} c_j + \frac{K - \sum_{j=1}^{r} a_j}{a_{r+1}} c_{r+1}. \tag{1.10}$$

Proof: Using the number r given above let us consider the following problem:

$$\max \left\{ \sum_{j=1}^{r} c_j y_j + \left(K - \sum_{j=1}^{r} a_j \right) \frac{c_{r+1}}{a_{r+1}} y'_{r+1} + \sum_{j=r+1}^{n} c_j y_{j+1} \right\} \tag{1.11}$$

$$\sum_{=1}^{r} a_j y_j + \left(K - \sum_{j=1}^{r} a_j \right) y'_{r+1} + \sum_{j=r+1}^{n} a_j y_{j+1} \leqq K$$

$$y_j \in \{0, 1\} \quad (j = 1, 2, \ldots, n + 1).$$

Problem (1.11) is obtained from (1.6) by adding a new variable after variable y_r. The ordering (1.7) still holds, furthermore condition (1.8) is satisfied for this enlarged problem, so Lemma 1.1 is applicable. Let us denote the set of feasible solutions of (1.11) by Q and the vector of objective function coefficients by $\mathbf{d}$. It then follows from Lemma 1.1, that

$$\sum_{j=1}^{r} c_j + \left(K - \sum_{j=1}^{r} a_j\right) \frac{c_{r+1}}{a_{r+1}} = \max_{y \in Q} \mathbf{d}^T \mathbf{y} \geqq \max_{y \in Q,\, y_{r+1}=0} \mathbf{d}^T \mathbf{y} = \max_{x \in R} \mathbf{c}^T \mathbf{x}$$

which proves the present lemma.

1.3 The cargo loading problem

A similar problem of two constraints is called the cargo loading problem. We are given the weight and volume capacity of a transport vehicle, e.g. a cargo boat, and the data of the indivisible objects (e.g. machines) that can be carried by the vessel. These data include the weight and volume of each object and also their value, which may be a real value, a use-value or simply the transportation cost depending on the nature of the problem. The freight of maximal value not exceeding the capacities is to be determined.

Notation

n	Number of different objects
a_j	Weight of object j
b_j	Volume of object j
c_j	Value of object j (see above)
K	Weight capacity of cargo boat
L	Volume capacity of cargo boat
r_j	Number of objects j available
x_j	Number of objects j to be transported (unknown)

The problem is formulated as follows

$$\begin{aligned} \max z &= \sum_{j=1}^{n} c_j x_j \\ &\sum_{j=1}^{n} a_j x_j \leqq K \\ &\sum_{j=1}^{n} b_j x_j \leqq L \qquad (1.12) \\ x_j &\in \{0, 1, 2, \ldots, r_j\} \quad (j = 1, 2, \ldots, n). \end{aligned}$$

In the practical cargo loading problem the coefficients a_j, b_j, c_j and the right hand sides K and L are obviously positive. In other problems it may not be so. Similarly to our previous discussion, the case $a_j, b_j, c_j \leqq 0$ can be reduced to the case of nonnegative coeffficients. Furthermore the variables having coefficients $c_j \geqq 0$, $a_j \leqq 0$, $b_j \leqq 0$ and $c_j \leqq 0$, $a_j \geqq 0$, $b_j \geqq 0$ can be ignored because there is an optimal solution with $x_j = r_j$ and $x_j = 0$, respectively. However, if the signs of a_j and b_j are different then there is no equivalent problem in general with nonnegative coefficients.

If all the coefficients are nonnegative, then the quantity

$$\min\left(\frac{K}{a_j}, \frac{L}{b_j}, r_j\right)$$

will obviously give a new upper bound for the value of variable x_j.

Without further restricting the generality, we can suppose that $K > 0, L > 0$, $a_j + b_j > 0$. If a_j (b_j) is equal to 0, then the ratio K/a_j (L/b_j) is considered $+\infty$.

If we take into account only one of the constraints, then – substituting the variables by those of 0–1 – we obtain a problem of the type described in the previous paragraph. If its optimal solution satisfies the other restriction, then the problem is solved. The situation is similar using linear combinations of the two constraints as a single restriction. It may happen however, that in no case will the abandoned constraint be satisfied. The solution of this particular problem and in general the solution of problems of several constraints will be discussed in later chapters.

1.4 The travelling salesman problem

The usual description of the problem is the following: A travelling salesman should visit n cities and return to the starting point (which is one of the n cities) at minimal cost. The travelling cost for the pairs of cities are given.

The importance of this problem lies in two facts. First of all, there are many practical problems that may be formulated in a similar way. For example, the distribution or collection of products; order of regular checking of several objects or monitoring of instruments; the order of jobs on a machine if the cost of changing from manufacturing one product to another is a function of these products (e.g. in a petroleum refinery). Secondly, the algebraic formulation of the problem seems to be difficult because of the extremely large number of constraints ensuring the existence of only one circuit. The inclusion of logical constraints, instead of algebraic ones, will lead to other difficulties. Thus, the travelling salesman problem is important in itself as well as being of interest to be used as a test problem.

Let us now look at three different models of the problem which appeared in Dantzig (1963).

1. Let the value of variable x_{ijt} be 1, if we go from city i to city j at step t and 0 otherwise. Obviously $x_{iit} = 0$ for all i and t. We have to ensure that every city is visited only once:

$$\sum_{j=1}^{n} \sum_{t=1}^{n} x_{ijt} = 1 \quad (i = 1, 2, \ldots, n). \tag{1.13}$$

Furthermore, we leave city j at step $t + 1$ if and only if we arrived there at step t.

$$\sum_{i=1}^{n} x_{ijt} = \sum_{k=1}^{n} x_{jk,\,t+1} \quad (j, t = 1, 2, \ldots, n). \tag{1.14}$$

As the end point is identical with the starting point

$$x_{ij,\,n+1} = x_{ij1}.$$

We have to find the values of the variables

$$x_{ijt} \in \{0, 1\} \quad (i, j, t = 1, 2, \ldots, n) \tag{1.15}$$

satisfying constraints (1.13) and (1.14) and minimizing the objective function

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{t=1}^{n} c_{ij} x_{ijt}. \tag{1.16}$$

2. Now one of the cities, say city 1, will have a special role. Namely, only tours containing city 1 will be accepted. Let $x_{ij} = 1$ or 0 according to whether the salesman goes from city i to city j or not. Now the problem can be formulated as

$$\begin{gathered}
\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\
\sum_{i=1}^{n} x_{ij} = 1 \quad (j = 1, 2, \ldots, n) \\
\sum_{j=1}^{n} x_{ij} = 1 \quad (i = 1, 2, \ldots, n) \\
u_i - u_j + n x_{ij} \leqq n - 1 \quad (2 \leqq i \neq j \leqq n) \\
x_{ij} \in \{0, 1\} \quad (i, j = 1, 2, \ldots, n)
\end{gathered} \tag{1.17}$$

where the variables u_i are arbitrary integer (or arbitrary real) numbers. The problem is similar to the well-known assignment problem, the only difference being the third group of constraints which ensure that only one round tour appears in the solution. In order to verify this, let us suppose that there are at least two round tours, in which case at least one of them does not contain city 1. Let the order of

cities in this tour be $i_1, i_2, \ldots, i_r, i_1$. Summing up the corresponding inequalities from the third group:

$$\begin{aligned} u_{i_1} - u_{i_2} + n &\leqq n - 1 \\ u_{i_2} - u_{i_3} + n &\leqq n - 1 \\ &\ldots \\ u_{i_{r-1}} - u_{i_r} + n &\leqq n - 1 \\ u_{i_r} - u_{i_1} + n &\leqq n - 1 \\ \hline rn &\leqq r(n-1) \end{aligned}$$

which is a contradiction. This proves that there is exactly one tour in the solution of (1.17). It remains for us to show that none of the full tours is excluded by the above constraints. Let us consider any single full tour and let us choose $u_i = t$, if city i is visited at step t. It is easy to see that all the constraints are satisfied.

3. The third formulation differs from the previous one only in the way of excluding the several round tours. First we solve the problem

$$\begin{aligned} &\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\ &\sum_{i=1}^{n} x_{ij} = 1 \quad (j = 1, \ldots, n) \qquad (1.18) \\ &\sum_{j=1}^{n} x_{ij} = 1 \quad (i = 1, \ldots, n) \\ &x_{ij} \in \{0, 1\} \quad (i, j = 1, \ldots, n) . \end{aligned}$$

If the solution consists of only one round tour, then the problem is solved. Otherwise let us consider any round tour:

$$i_1, i_2, \ldots, i_r, i_1 \quad r < n .$$

Then the following constraint will exclude the present, non feasible solution:

$$x_{i_1 i_2} + x_{i_2 i_3} + \ldots + x_{i_{r-1} i_r} + x_{i_r i_1} \leqq r - 1 . \qquad (1.19)$$

Now let us solve problem (1.18)–(1.19). If the solution still consists of more than one round tour we form again a constraint of type (1.19). The process is continued until the desired one round tour solution is obtained – naturally in a finite number of steps.

As the first model contains n^3 variables, it is not practical for larger problems. In the second model the number of constraints is of the order of n^2, which is too large. In the third model we are concerned about having a long series of problems. Later we shall see that the modification of this model – introduction of logical constraints – will help to solve this problem.

A good survey of algorithms is to be found in Bellmore and Nemhauser (1968)

1.5 Fixed-charge problems

In practice, there are problems that can be formulated as linear programming problems with an additional cost for each product. This fixed charge is independent of the quantity produced (and is also independent of the development of the new product, new machines, training of workers, advertising, etc.). Naturally if any of the products is not produced, then the corresponding fixed-charge will not be levied.

Notation

$A = \{a_{ij}\}$	Matrix of technological coefficients
a_{ij}	Quantity of resource i needed for producing one unit of product j
b_i	Quantity of resource i available
c_j	Profit on one unit of product j
d_j	Fixed-charge of product j
x_j	Quantity of product j to be produced

The problem of maximizing the net profit is formulated as follows:

$$\begin{aligned} &\max \mathbf{c}^{\mathrm{T}}\mathbf{x} - \delta^{\mathrm{T}}(\mathbf{x})\mathbf{d} \\ &\mathbf{A}\mathbf{x} \leqq \mathbf{b} \\ &\mathbf{x} \geqq \mathbf{0} \end{aligned} \tag{1.20}$$

where

$$\delta_j(x) = \begin{cases} 0, & \text{if } x_j = 0 \\ 1, & \text{if } x_j > 0 \end{cases} \tag{1.21}$$

and it is supposed that $\mathbf{d} \geqq \mathbf{0}$.

Denoting the unit matrix by $\mathbf{E}$, let us consider the following convex polyhedron (the set of feasible solutions of (1.20)):

$$\mathbf{R} = \{\mathbf{x} \mid \hat{\mathbf{A}}\mathbf{x} \leqq \hat{\mathbf{b}}\} \tag{1.22}$$

where

$$\hat{\mathbf{A}} = \begin{bmatrix} \mathbf{A} \\ -\mathbf{E} \end{bmatrix}, \quad \hat{\mathbf{b}} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix}.$$

Problem (1.20) is a mixed variable problem. In principle, any of the algorithms for mixed problems (discussed later), may be applied.

Heuristic algorithms may be constructed for (1.20) with the help of the following lemma.

Lemma 1.3: *The function*

$$\mathbf{c}^{\mathrm{T}}\mathbf{x} - \delta^{\mathrm{T}}(\mathbf{x})\mathbf{d}$$

takes its global maximum on the set R at a vertex of the convex polyhedron R.

The lemma will be proved in a more general form.

Lemma 1.4: *If the convex function $f(\mathbf{x})$ takes its global maximum on the set R then it is also taken at a vertex of the convex polyhedron R.*

Proof of Lemma 1.4: It is sufficient to show that if $x' \in R$ is a global maximum of function $f(\mathbf{x})$ on the set R, and $\mathbf{x}'$ is not a vertex of polyhedron R, then there exists another point $\mathbf{x}'' \in R$ such that

$$f(\mathbf{x}'') = f(\mathbf{x}'),$$

furthermore, point $\mathbf{x}''$ lies on at least one more hyperplane defined by the equations

$$H_i = \{\mathbf{x} \mid \hat{\mathbf{a}}^i \mathbf{x} = \hat{b}_i\}$$

where $\hat{\mathbf{a}}^i$ is the ith row of matrix $\hat{\mathbf{A}}$. Furthermore let us define the manifold Q as the intersection of hyperplanes containing point $\mathbf{x}'$:

$$Q = \bigcap_{\mathbf{x}' \in H_i} H_i .$$

If x' lies in none of the hyperplanes H_i, then let Q be the entire n-dimensional Euclidean space.

Then $R \cap Q$ is also a polyhedron. It will be shown that x' is not a vertex of this polyhedron. As x' is not a vertex of polyhedron R according to our supposition, there exist two points $\mathbf{x}^1, \mathbf{x}^2 \in R$ such that

$$\mathbf{x}' = \lambda \mathbf{x}^1 + (1 - \lambda)\mathbf{x}^2 \quad (0 < \lambda < 1)$$

It is sufficient to show, that $\mathbf{x}^1, \mathbf{x}^2 \in Q$.

If Q is the entire Euclidean space, then this is trivial. Otherwise let us suppose indirectly that $\mathbf{x}^1 \notin Q$, that is, there exists a hyperplane H_r such that

$$\mathbf{x}' \in \mathrm{H}_r \quad \text{and} \quad \mathbf{x}^1 \notin H_r .$$

Then the equation

$$\hat{\mathbf{a}}^r \mathbf{x}' = \hat{b}_r$$

implies that $\hat{\mathbf{a}}^r \mathbf{x}^1 < \hat{b}_r$ and $\hat{\mathbf{a}}^r \mathbf{x}^2 > \hat{b}_r$ or vice versa.

But this contradicts the supposition that $\mathbf{x}^2 \in R$, thus both points $\mathbf{x}^1$ and $\mathbf{x}^2$ must be on the variety Q.

The convex polyhedron R does not contain a complete straight line therefore there exists a point

$$\mathbf{x}^5 = \lambda \mathbf{x}^1 + (1 - \lambda)\mathbf{x}^2 \quad (\lambda < 0 \quad \text{or} \quad \lambda > 1)$$

for which at least one constraint is violated

$$\hat{\mathbf{a}}^k \mathbf{x}^5 > \hat{b}_k .$$

As $\mathbf{x}^1 \in R$ and $\mathbf{x}^5 \notin R$, there is a point

$$\mathbf{x}^3 = \mu \mathbf{x}^1 + (1 - \mu)\mathbf{x}^5 \quad (0 < \mu < 1) \quad \text{and} \quad \hat{\mathbf{a}}^k \mathbf{x}^3 = \hat{b}_k .$$

The point lies on the straight line defined by points $\mathbf{x}^1$ and $\mathbf{x}^2$, thus $\mathbf{x}^1, \mathbf{x}^2 \in Q$ implies $\mathbf{x}^3 \in Q$. Therefore $\mathbf{x}^3$ lies on at least one more hyperplane H_i than does $\mathbf{x}'$

It remains for us to show that function $f(\mathbf{x})$ takes its global maximum at this point, that is,

$$f(\mathbf{x}^3) = f(\mathbf{x}').$$

For this purpose, let us denote by $\mathbf{x}^4$ the point $\mathbf{x}^1$ or $\mathbf{x}^2$ in such a way that the interval $(\mathbf{x}^3, \mathbf{x}^4)$ should contain point $\mathbf{x}'$

$$\mathbf{x}' = \mu\mathbf{x}^3 + (1 - \mu)\mathbf{x}^4 \quad (0 < \mu < 1).$$

As $f(\mathbf{x}')$ is a global maximum

$$f(\mathbf{x}^3) \leqq f(\mathbf{x}'), \; f(\mathbf{x}^4) \leqq f(\mathbf{x}'),$$

but

$$f(\mathbf{x}^3) < f(\mathbf{x}')$$

is impossible because, then

$$f(\mu\mathbf{x}^3 + (1 - \mu)\mathbf{x}^4) > \mu f(\mathbf{x}^3) + (1 - \mu)f(\mathbf{x}^4)$$

which contradicts the convexity of function $f(\mathbf{x})$ and completes the proof of Lemma 1.4.

Proof of Lemma 1.3: Using Lemma 1.4, it is sufficient to show that the function

$$\mathbf{c}^T\mathbf{x} - \delta^T(\mathbf{x})\mathbf{d}$$

is convex, that is, the function

$$\delta^T(\mathbf{x})\mathbf{d}$$

is concave. As the sum of concave functions is also concave and

$$\delta^T(\mathbf{x})\mathbf{d} = \sum_{j=1}^{n} \delta_j(x_j)d_j,$$

it is sufficient to see that $\delta_j(x_j)$ is concave, which is trivial.

Using the result of Lemma 1.4 in searching for the optimal solution of (1.20), we can confine ourselves to seeking only the vertices of polyhedron R. But in the case of larger problems the number of vertices may be extremely high, therefore the need to examine all vertices is not a feasible approach for solving this problem.

1.6 An investment problem

The most important applications of discrete programming are the different kinds of investment problems. Let us consider now the following simple combined production–investment problem. At the beginning, we have certain basic resources

and production capacities. These may be augmented by realizing some of the given investments. The maximal profit and the production plan as well as the necessary investments are to be determined. There is a limit to the amount of money used for the investments. This limit may be treated as a parameter.

Notation

a_{ij} Quantity of resource (or production capacity) i needed to produce one unit of product j
b_0 Total amount of money available for investment
b_i Quantity of resource i available before realizing the investments
c_j Net profit on one unit of product j
d_k Cost of investment k
g_{ik} Quantity of additional resource i, if investment k is realized
x_k 1 or 0 depending on whether investment k is realized or not
y_j Quantity of product j to be produced

Introducing the short **notation** for the coefficients,

$$\mathbf{c}^T = (c_1, \ldots, c_q) \qquad \mathbf{G} = \begin{pmatrix} g_{11} \ldots g_{1p} \\ \vdots \\ g_{m1} \ldots g_{mp} \end{pmatrix} \qquad \mathbf{A} = \begin{pmatrix} a_{11} \ldots a_{1q} \\ \vdots \\ a_{m1} \ldots a_{mq} \end{pmatrix}$$
$$\mathbf{b}^T = (b_1, \ldots, b_n)$$
$$\mathbf{d}^T = (d_1, \ldots, d_p)$$

the problem can be formulated as follows

$$\begin{aligned} &\max \mathbf{c}^T\mathbf{y} \\ &\mathbf{d}^T\mathbf{x} \leqq b_0 \\ &\mathbf{A}\mathbf{y} - \mathbf{G}\mathbf{x} \leqq \mathbf{b} \\ &x_k \in \{0, 1\} \quad (k = 1, \ldots, p) \\ &y_j \geqq 0 \quad (j = 1, \ldots, q). \end{aligned} \tag{1.23}$$

If this problem is solved for different values of b_0 the corresponding relative efficiency can be calculated and a better investment policy can be obtained. Investments of higher complexity (e.g. decisions for several time periods) can also be taken into account by similar models.

Problem (1.23) is a special integer programming problem.

If only given production capacities (h_i) are to be set up at a minimal cost, then by introducing the demand vector

$$\mathbf{h}^T = (h_1, \ldots, h_m)$$

we obtain the following simple pure integer programming problem:

$$\begin{aligned} &\min \mathbf{d}^T\mathbf{x} \\ &\mathbf{Gx} \geqq \mathbf{h} \\ &x_j \in \{0, 1\} \quad (j = 1, \ldots, p). \end{aligned} \tag{1.24}$$

1.7 A plant location problem

This is also an investment problem, but it is more difficult than those mentioned above. It is more difficult not only because the investment costs are taken into account but because there are other costs depending on the location of plants with respect to each other, e.g. transport costs. Another difficulty is that the investment costs are paid once only but the transport costs occur periodically. In order to make them comparable the investment costs and the transport costs of several periods are added, naturally the costs for the latter are discounted for the first period. In other words, a suitable constant multiple of transport costs is considered.

Notation

K Number of sites suitable for building plants
L Number of different plants
J Number of different products to be produced
a_{ir} Quantity of product i produced (consumed if $a_{ir} < 0$) by plant r.
b_i Demand of product i (requirements of plants to be located is not counted here)
c_{pr} Cost of location for plant r at place p
d_{pq} Cost of transporting one unit between places p and q.
f_{rs} Intensity of transport (quantity demanded in one period) from plant r to plant s
λ Multiplier for transport costs, making them comparable to investment costs (see above).
x_{pr} Location variable, 1 or 0 depending on whether plant r is located at place p or not.

For the sake of simplicity, it is supposed that only one plant may be located at one place. The quantity b_i may be negative, in this case the system of plants may have a demand of at most $-b_i$ on product i from outer sources.

Now we formulate a model on the basis of which the locations of plants can be determined. The production requirements should be satisfied at a minimal total cost

$$\min \sum_{p=1}^{K} \sum_{r=1}^{L} (c_{pr} + \lambda \sum_{q=1}^{K} \sum_{s=1}^{L} d_{pq} f_{rs} x_{qs}) x_{pr} \tag{1.25}$$

subject to

$$\sum_{p=1}^{K} \sum_{r=1}^{L} a_{ir} x_{pr} \geqq b_i \quad (i = 1, \ldots, M)$$

$$\sum_{r=1}^{L} x_{pr} \leqq 1 \quad (p = 1, \ldots, K) \tag{1.26}$$

$$x_{pr} \in \{0, 1\} \quad (p = 1, \ldots, K; \quad r = 1, \ldots, L).$$

This is a nonlinear discrete programming problem. It can be solved by a version of the method described in Chapter 3.

If we do not wish to or cannot determine which individual plant satisfies the consumption of a particular plant, then the problem (1.25)–(1.26) may be combined with a transport problem.

Another version of the problem may be used for analysing the economic efficiency of similar plants of different size.

1.8 The empty container problem

This problem is similar to that of cargo loading, it is only the point of view that is different. Now, we wish to carry all the given objects using a minimum number of containers. Notation as in Section 1.3 except for variable x_j.

x_{ij} Number of objects j carried in container i
$m + 1$ An upper bound for the number of containers used.

In other words, $m + 1$ containers can certainly hold all the objects. It is easy to get such an upper bound by loading the objects in any order one after the other until either the weight or the volume capacity of the container is used up. Then the next container is taken, and so on. A preordering of objects may result in a better starting solution, i.e. a smaller upper bound m. Some heuristics may also be used to improve this starting solution, e.g. certain changes between containers one of which has used up its full weight capacity, the other its full volume capacity and their other capacities are largely unused. The importance of these improvements lies in the fact that the size of the problem depends on the bound m. Furthermore, let

$$y_i = \begin{cases} 1, \text{ if container } i \text{ is used} \\ 0 \text{ otherwise.} \end{cases}$$

We want to minimize the number of containers used:

$$\min \sum_{i=1}^{m} y_i \tag{1.27}$$

subject to

$$\sum_{j=1}^{n} a_j x_{ij} \leqq K y_i \qquad (i = 1, \ldots, m)$$

$$\sum_{j=1}^{n} b_j x_{ij} \leqq L y_i \qquad (i = 1, \ldots, m)$$

$$\sum_{i=1}^{m} x_{ij} = r_j \qquad (j = 1, \ldots, n) \tag{1.28}$$

$$x_{ij} \in \{0, 1, \ldots, r_j\} \qquad (i = 1, \ldots, m; j = 1, \ldots, n)$$

$$y_i \in \{0, 1\} \qquad (i = 1, \ldots, m).$$

The first two groups of constraints (1.28) express that the capacity requirements must be fulfilled if the container is in use: $y_i = 1$, otherwise it must remain empty. The third group means that all the available objects must be carried.

During the process of solving the problem, m is always one unit smaller than the best objective function value known up to now, because we are interested solely in better solutions.

In many of the algorithms which will be discussed later the number of feasible solutions largely influences the efficiency of the algorithm. From this point of view the formulation of requirements in the form (1.28) is not very suitable. If, for example, two feasible solutions differ only in the order of containers then – as they are not substantially different solutions of the problem – it would be sufficient to permit only one of them. For this purpose we introduce two further constraints:

$$x_{11} \geqq x_{21} \geqq \ldots \geqq x_{m1} \tag{1.29}$$

and the equations

$$x_{i1} = x_{i+1,1}; \; x_{i2} = x_{i+1,2}; \ldots x_{ir} = x_{i+1,r} \tag{1.30a}$$

should imply, that

$$x_{i,r+1} \geqq x_{i+1,r+1} \tag{1.30b}$$

for all i and r.

In any feasible solution of the problem, if a container is empty, then all the containers, having higher serial numbers must also be empty.

The constraints (1.29) and (1.30) mean nothing other than the introduction of lexicographical ordering, which is discussed in Chapter 2.

The problem is now formulated as minimizing function (1.27), subject to the constraints (1.28)–(1.30).

In the remaining section of the present chapter the reduction of some mathematical programming problems to discrete programming problems will be discussed briefly, mainly on the basis of Dantzig (1963) where further details may be found.

1.9 Alternative constraints

Given two functions $G(\mathbf{x})$ and $H(\mathbf{x})$ lower bounded on the set R. If at least one of the constraints

$$G(\mathbf{x}) \geqq 0 \tag{1.31}$$

$$H(\mathbf{x}) \geqq 0 \tag{1.32}$$

must hold at each point of the set R, then this may be reformulated by introducing a discrete variable. Let us denote the lower bounds of the functions $G(\mathbf{x})$ and $H(\mathbf{x})$ on the set R by L_G and L_H, respectively

$$G(\mathbf{x}) \geqq L_G,\ H(\mathbf{x}) \geqq L_H \text{ for all } \mathbf{x} \in R. \tag{1.33}$$

Then the constraints (1.31) and/or (1.32) are satisfied if and only if

$$\begin{aligned} G(\mathbf{x}) - \delta L_G &\geqq 0 \\ H(\mathbf{x}) - (1-\delta)L_H &\geqq 0 \\ \delta &\in \{0, 1\}. \end{aligned} \tag{1.34}$$

This reformulation may be very useful if the number of dichotomies (k) is high, so that the solution of the corresponding 2^k problems is unrealistic.

Example 1: In the case of complementary variables (e.g. in quadratic programming) the constraint

$$\sum_{j=1}^{n} x_j y_j = 0, \quad x_j \geqq 0, \quad y_j \geqq 0 \quad (j = 1, \ldots, n). \tag{1.35}$$

may be substituted by n dichotomies:

$$-x_j \geqq 0 \quad \text{or} \quad -y_j \geqq 0 \quad (j = 1, \ldots, n). \tag{1.36}$$

Example 2: In production problems, it is often required that the produced quantity must be over a certain limit if it is manufactured at all. Supposing that the quantity must be nonnegative, the dichotomy

$$x_1 - K \geqq 0 \quad \text{or} \quad -x_1 \geqq 0$$

will do the job, where K is the permitted minimal production level.

Example 3: Optimization over nonconvex regions.

Let $$T = \bigcup_{i=1}^{m} R_i,$$

where $$R_i = \{\mathbf{x} \mid G_{ij}(\mathbf{x}) \geqq 0, \quad (j = 1, \ldots, n)\}$$

and the functions $G_{ij}(\mathbf{x})$ are concave, which implies that the sets R_i are also convex. Furthermore, let

$$G_{ij}(\mathbf{x}) \geqq L_{ij} \quad \text{for all} \quad x \in T' \geq T.$$

Then T is the set of points x satisfying the following constraints:

$$G_{ij}(\mathbf{x}) \geqq \delta_i L_{ij} \qquad (i = 1, \ldots, m)$$
$$(j = 1, \ldots, n)$$
$$\sum_{i=1}^{m} \delta_i \leqq m - 1$$
$$\delta_i \in \{0, 1\} \qquad (i = 1, \ldots, m)\,.$$

1.10 Optimization of nonlinear separable functions

If the maximum or minimum of a function of the type

$$\sum_{j=1}^{n} \phi_j(x_j)$$

is to be determined on a given domain, then the function may be linearized by introducing discrete variables. Let us consider only one member of the above sum without a subscript. The function $\phi(x)$ of one variable can be approximated by a piecewise linear function in the following way. Let

$$\phi(a_i) = b_i \qquad (i = 0, 1, \ldots, k)$$

where $a_0, a_1, \ldots, a_k$ are the dividing points of a sufficiently dense subdivision of the domain of variable x, determined by the original problem. Using the short notation

$$h_i = a_i - a_{i-1}$$

and

$$s_i = \frac{b_i - b_{i-1}}{a_i - a_{i-1}}\,.$$

Then

$$\phi(x) = b_0 + \sum_{i=1}^{k} s_i y_i$$
$$x = \sum_{i=1}^{k} y_i$$
$$0 \leqq y_i \leqq h_i \qquad (i = 1, \ldots, k)$$
$$y_i - h_i \geqq 0 \quad \text{or} \quad -y_{i+1} \geqq 0 \qquad (i = 1, \ldots, k-1)\,.$$

The last pair of alternative constraints ensures that the variable y_{i+1}, representing the $(i + 1)$st interval, should remain zero unless the previous interval is already filled, that is, $y_i = h_i$. According to the previous section this may be expressed as

$$y_i - h_i = -\delta_i h_i$$
$$-y_i = -(1 - \delta_i) h_{i+1} \qquad (i = 1, \ldots, k-1)$$
$$\delta_i \in \{0, 1\}\,.$$

1.11 Conditional constraints

Let us consider an optimization problem on a set R. If the constraint

$$H(\mathbf{x}) \geqq 0$$

is required only if

$$G(\mathbf{x}) > 0,$$

then this may be expressed as the following dichotomy

$$G(\mathbf{x}) > 0,\ H(\mathbf{x}) \geqq 0$$

or

$$G(\mathbf{x}) \leqq 0\,.$$

This is similar to the problem discussed in Section 1.9. The only difficulty is that of handling the strict inequality, but it can be avoided as follows. Let L_H; L_G and U_G be lower and upper bounds of the functions $H(\mathbf{x})$ and $G(\mathbf{x})$ respectively on the set R

$$\begin{aligned} U_G \geqq G(\mathbf{x}) \geqq L_G \quad &\text{for any} \quad \mathbf{x} \in R \\ H(\mathbf{x}) \geqq L_H \quad &\text{for any} \quad \mathbf{x} \in R\,. \end{aligned}$$

Then the above pair of alternative constraints may be reformulated as

$$\begin{aligned} &H(\mathbf{x}) \geqq \delta L_H \\ &G(\mathbf{x}) \leqq (1 - \delta)U_G \\ &\delta \in \{0, 1\}\,. \end{aligned}$$

1.12 Minimization of concave functions on a convex polyhedron

A concave function $f(\mathbf{x})$ is to be minimized on the convex polyhedron R. According to Lemma 1.4, if the function $f(\mathbf{x})$ is continuous, concave and bounded from below on the convex polyhedron R, then there is a vertex of polyhedron R where the global minimum of function $f(\mathbf{x})$, on the set R is reached.

Let us suppose that the function $f(\mathbf{x})$ of n variables is approximated from below by the hyperplanes

$$\mathbf{a}^i\mathbf{x} = b_i \quad (i = 1, 2, \ldots, p)$$

where the vector $\mathbf{a}^i$ is a row vector, that is,

$$0 \leqq \min_i \{\mathbf{a}^i\mathbf{x} - b_i\} - f(\mathbf{x}) \leqq \varepsilon$$

where ε is the desired accuracy.

The original problem

$$\min \{f(\mathbf{x}) \mid \mathbf{x} \in R\}$$

may be formulated as

$$\min \{z \mid z - f(\mathbf{x}) \geqq 0, \quad \mathbf{x} \in R\}.$$

Substituting the function $f(\mathbf{x})$ by the approximating expression

$$\min_i \{\mathbf{a}^i x - \mathbf{b}_i\},$$

the problem becomes

$$\min \{z \mid (\mathbf{x}, z) \in T\}$$

where

$$T = \{(\mathbf{x}, z) \mid \exists \mathrm{i}: z - (\mathbf{a}^i \mathbf{x} - b_i) \geqq 0 \quad 1 \leqq i \leqq k\}.\dagger$$

Let M be an upper bound of the differences of approximating linear functions on the polyhedron R

$$(\mathbf{a}^i \mathbf{x} - b_i) - (\mathbf{a}^k \mathbf{x} - b_k) \leqq M \quad \text{for any } i, k = 1, \ldots, p \text{ and } \mathbf{x} \in R.$$

Then the points of set T are the solutions of the following system of inequalities

$$\begin{gathered} z - (\mathbf{a}^i \mathbf{x} - b_i) \geqq -M(1 - \delta_i) \quad (i = 1, \ldots, p) \\ \delta_1 + \delta_2 \ldots + \delta_p \geqq 1 \\ \delta_i \in \{0, 1\} \quad (i = 1, \ldots, p) \\ \mathbf{x} \in R. \end{gathered}$$

Thus the problem is reduced to another one of mixed integer type.

† The sign ∃ means: there exists.

CHAPTER 2

SINGLE BRANCH IMPLICIT ENUMERATION METHODS

Enumeration methods play an important role in solving integer programming problems.† Their substance is to enumerate all solutions†† i.e. all integer vectors in the given range and to choose the best one according to the objective function among those satisfying the constraints of the problem. However, complete enumeration (the explicit examination of all solutions) is not possible even for problems of moderate size. The enumeration becomes a method only if a high proportion of solutions is implicitly examined, i.e. excluded in large groups without direct checking, as either they are shown to violate the set of constraints or it is made obvious that they cannot have a better objective function value than that of the best feasible solution found so far. The methods described and summarized in the present chapter have a common property. The solutions are explicitly or implicitly examined either in a predetermined order or in an order gradually formed by the algorithm.

In other words we never have to return to any one group of solutions after examining another. For this reason these algorithms have acquired the descriptive name, as in the title of this chapter: single branch implicit enumeration methods. These methods contrast with those described in Chapters 3 and 4.

2.1 Partial and lexicographic ordering of vectors

In the enumeration algorithms we shall need the following two orderings of the n-dimensional Euclidean space, E^n.

For any two vectors $\mathbf{x}, \mathbf{y} \in E^n$ the *partial ordering relation*

$$\mathbf{x} \leqq \mathbf{y}$$

holds, if

$$x_j \leqq y_j \quad j = (1, \ldots, n). \tag{2.2}$$

† For simplicity, in this chapter only pure discrete problems are considered.

†† In integer programming a numerical vector of appropriate size usually is called the solution of a given problem if it satisfies the integer requirements and the bounds which are given for each of the discrete variables. If all constraints of the problem are satisfied, then the vector is a feasible solution of the problem. We shall adopt this convention.

Note: This relation is called partial ordering because not all pairs of n-dimensional vectors may be ordered in this way. For example, if

$$\mathbf{x} = (1, 2) \qquad \mathbf{y} = (3, 1)$$

then neither $\mathbf{x} \leqq \mathbf{y}$ nor $\mathbf{y} \leqq \mathbf{x}$. Although many other possible partial orderings could have been defined, in the present book we shall use only this one. Whenever we use the expression "partial ordering" the above-defined partial ordering is understood. This also means that the use of $\leqq$ between two vectors cannot be misleading as it always signifies this relation.

Now we define also a full ordering.

For any two vectors $\mathbf{x}, \mathbf{y} \in E^n$, vector $\mathbf{x}$ precedes vector $\mathbf{y}$ according to the *lexicographic ordering*, that is,

$$\mathbf{x} \prec \mathbf{y}, \tag{2.3}$$

if there exists a subscript $1 \leqq r \leqq n$, such that

$$x_j = y_j \quad \text{for any} \quad j < r \tag{2.4}$$

and

$$x_r < y_r . \tag{2.5}$$

Obviously any two different vectors of the same dimension may be ordered according to the lexicographic ordering simply because the first different components determine the order.

Example: If $\mathbf{x} = (1, -2, 3)$ and $\mathbf{y} = (1, -3, 7)$ then $\mathbf{y} \prec \mathbf{x}$.

Note 1: There are two different ways of defining orderings of a set according to whether an element is in relation with itself or not. The above two orderings fall into different categories from this point of view in accordance with the usual notation and usage. It is very easy to see that these orderings satisfy the corresponding axioms, i.e. they may be called orderings.

Note 2: Confining ourselves to the set

$$D^n = \{\mathbf{x} \mid \mathbf{x} \in E^n, \quad x_j \in \{0, 1\} \quad (i = 1, \ldots, n)\} \tag{2.6}$$

the lexicographic ordering may be defined in a different way.

An integer

$$s(\mathbf{x}) = x_1 2^{n-1} + x_2 2^{n-2} + \ldots . + x_n 2^0 \tag{2.7}$$

is assigned to each vector $\mathbf{x} = (\mathbf{x}_1, \ldots . , \mathbf{x}_n)$. Then vector $\mathbf{x}$ precedes vector $\mathbf{y}$

$$\mathbf{x} \prec \mathbf{y}, \quad \text{if and only if} \quad s(x) < s(y) . \tag{2.8}$$

This ordering is also called numerical ordering. The number $s(x)$ also gives the position of vector $\mathbf{x}$ in this ordering, apart from the vector $\mathbf{x} = \mathbf{0}$. This definition

may be applied for any finite subset of E^n using an appropriate base instead of 2 in expression (2.7).

Example: Enumerate all the three-dimensional vectors of 0–1 component, in lexicographic ordering. The solution is (columnwise):

$$\begin{array}{ll} (0,0,0) & (1,0,0) \\ (0,0,1) & (1,0,1) \\ (0,1,0) & (1,1,0) \\ (0,1,1) & (1,1,1). \end{array}$$

2.2 The Lawler–Bell algorithm

In the present and following sections a method will be discussed which is due to Lawler and Bell (1966). Let us consider the following problem:

$$\min g_0(\mathbf{x}) \tag{2.9}$$

subject to

$$g_{i1}(\mathbf{x}) - g_{i2}(\mathbf{x}) \geqq 0 \quad (i = 1, \ldots, m) \tag{2.10}$$

$$x_j \in \{0, 1\} \quad (j = 1, \ldots, n) \tag{2.11}$$

where

$$\mathbf{x} = (x_1, \ldots, x_n)$$

and the functions

$$g_0(\mathbf{x}), \quad g_{i1}(\mathbf{x}), \quad g_{i2}(\mathbf{x}) \quad (i = 1, \ldots, m)$$

are monotone nondecreasing. Obviously the following two definitions are equivalent.

Definition 2.1: *A function $f(\mathbf{x})$ of n variables is monotone nondecreasing on a set $S \subset E^n$, if the function $f(x_1^*, \ldots, x_{k-1}^*, x_k, x_{k+1}^*, \ldots, x_n^*)$ of one variable is monotone nondecreasing on the set $\{\mathbf{x} \mid (x_1^*, \ldots, x_{k-1}^*, x_k, x_{k+1}^*, \ldots, x_n^*) \in S\}$* for any k and for *any fixed values of $x_1^*, \ldots, x_{k-1}^*, x_{k+1}^*, \ldots, x_n^*$.*

Definition 2.2: *A function $f(\mathbf{x})$ of n variables is monotone nondecreasing on a set $S \subset E^n$, if $\mathbf{x} \leqq \mathbf{y}$ implies $f(\mathbf{x}) \leqq f(\mathbf{y})$ for any $\mathbf{x}, \mathbf{y} \in S$.* In our case $S = D^n$ is the set of n-dimensional vectors with components 0 or 1, that is,

$$D^n = \{\mathbf{x} \mid x \in E^n, x_j \in \{0, 1\} \quad (j = 1, \ldots, n)\}.$$

Now all solutions of problem (2.9)–(2.11), that is, the elements of D^n, will be enumerated in lexicographic order. In other words, they will be substituted one after another for the constraints (2.10) and the best feasible solution found so far is always saved. We shall hopefully take large steps however using the result of

the following Lemma 2.1. Let us define two vectors $\mathbf{x}^{*-}$, $\mathbf{x}^* \in D^n$ for each solution x, that is, to the elements of D^n by the following properties:

(a) $\mathbf{x} \preceq \mathbf{x}^{*-} \prec \mathbf{x}^*$

(b) $\mathbf{x} \preceq \tilde{\mathbf{x}} \preceq \mathbf{x}^{*-}$ implies $\mathbf{x} \leqq \tilde{\mathbf{x}}$

(c) $\mathbf{x} \not\preceq \mathbf{x}^*$

(d) $s(\mathbf{x}^*) = s(\mathbf{x}^{*-}) + 1$,

where the function $s(x)$ is defined by expression (2.7). In other words, $\mathbf{x}^{*-}$ is the last vector in the numerical ordering among those that are greater than or equal to $\mathbf{x}$ also in the partial ordering $\mathbf{x}^{*-}$ is uniquely defined. $\mathbf{x}^*$ is the vector which immediately follows $\mathbf{x}^{*-}$ in the numerical ordering.† If such pair of vectors $\mathbf{x}^{*-}$ and $\mathbf{x}^*$ does not exist, then, in accordance with the verbal description, we define only

$$\mathbf{x}^{*-} = (1, 1, \ldots, 1)$$

the last vector in numerical ordering.

Let us choose any vector $\mathbf{x} \in D^n$, for which the set

$$D^n(\mathbf{x}) = \{\mathbf{y} \mid \mathbf{y} \in D^n, s(\mathbf{y}) < s(\mathbf{x}), g_{i1}(\mathbf{y}) - g_{i2}(\mathbf{y}) \geqq 0 \quad (i = 1, 2, \ldots, m)\}$$

is nonempty. Then one of the vectors satisfying the following two relations will be denoted by $\hat{\mathbf{x}}$

$$\hat{\mathbf{x}} \in D^n(\mathbf{x})$$

and

$$g_0(\hat{\mathbf{x}}) = \min \{g_0(\mathbf{y}) \mid \mathbf{y} \in D^n(\mathbf{x})\}.$$

The following lemma is also valid if $D^n(\mathbf{x})$ is empty. In this case $g_0(\hat{\mathbf{x}})$ is considered as $+\infty$.

Lemma 2.1: *No vector* $\tilde{\mathbf{x}}$, $\mathbf{x} < \tilde{\mathbf{x}} \leqq \mathbf{x}^{*-}$ *can be a feasible solution of problem* (2.9)–(2.11), *and better than the best feasible solution of the set* $D^n(\mathbf{x}) \cup \{\mathbf{x}\}$ *in any of the following cases:*

1° *if* $g_0(\mathbf{x}) \geqq g_0(\hat{\mathbf{x}})$

2° $\mathbf{x}$ *is a feasible solution, i.e. it satisfies the constraints* (2.10), (2.11)

3° *if for at least one of the subscripts* $i = 1, \ldots, m$ $g_{i1}(\mathbf{x}^{*-}) - g_{i2}(\mathbf{x}) < 0$.

Proof: Take any $\tilde{\mathbf{x}}$ ($\mathbf{x} \leqq \tilde{\mathbf{x}} \leqq \mathbf{x}^{*-}$). The statements of the lemma may be shown as follows:

1° As the function $g_0(\mathbf{x})$ is monotone nondecreasing, $g_0(\tilde{\mathbf{x}}) \geqq g_0(\mathbf{x}) \geqq g_0(\hat{\mathbf{x}})$, thus $\tilde{\mathbf{x}}$ cannot be better than $\hat{\mathbf{x}}$.

† An alternative definition for $\mathbf{x}^{*-}$ and $\mathbf{x}^*$ which is suitable for hand calculations is the following: Let us find the last ordered 0, 1 pair of neighbouring components. If there is no such a pair, then $\mathbf{x}^{*-} = (1, 1, \ldots, 1)$ and $\mathbf{x}^*$ is not defined. In the opposite case $\mathbf{x}^{*-}$ is obtained from $\mathbf{x}$ substituting by 1 all elements following the above defined pair. $\mathbf{x}^*$ is defined as the vector immediately following $\mathbf{x}^{*-}$ in the numerical ordering, that is, $s(\mathbf{x}^*) = s(\mathbf{x}^{*-}) + 1$. The equivalence of the two definitions will be shown in Lemma 2.2.

2° It is sufficient to consider the case

$$g_0(\mathbf{x}) < g_0(\hat{\mathbf{x}}),$$

because the opposite one was covered in 1°. Then $\mathbf{x}$ becomes the best feasible solution found up to now and as $g_0(\tilde{\mathbf{x}}) \geqq g_0(\mathbf{x})$ still holds, $\tilde{\mathbf{x}}$ cannot be better.

3° As the functions g_{i1} and g_{i2} are monotone

$$g_{i1}(\tilde{\mathbf{x}}) - g_{i2}(\tilde{\mathbf{x}}) \leqq g_{i1}(\mathbf{x}^{*-}) - g_{i2}(\mathbf{x}) < 0,$$

which means that $\tilde{\mathbf{x}}$ is not a feasible solution of problems (2.9)–(2.11).

The lemma implies that if any one of the conditions 1°–3° is satisfied, then the vector $\mathbf{x}$ may be followed by vector $\mathbf{x}^*$ in the process of solving the problem, or if it does not exist the algorithm is terminated.

It only remains to show an easy way of calculating $\mathbf{x}^*$ for a given vector $\mathbf{x} \in D^n$.

Lemma 2.2: *A vector $\mathbf{x} \in D^n$ is given.*

1° *If there exists a pair of indices, r, q, such that $2 \leqq r \leqq q \leqq n$ and*

$$x_{r-1} = 0, x_r = x_{r+1} = \ldots x_q = 1, \qquad (2.12)$$

$$x_{q+1} = x_{q+2} = \ldots = x_n = 0,$$

then the components of $\mathbf{x}^*$ may be written as

$$\begin{aligned} x_j^* &= x_j \quad j \leqq r-2 \\ x_{r-1}^* &= 1 \qquad (2.13) \\ x_j^* &= 0 \quad j \geqq r. \end{aligned}$$

2° *If such a pair of indices does not exist, then*

$$x^{*-} = (1, 1, \ldots, 1).$$

Proof 1°: Consider the vectors of D^n following vector $\mathbf{x}$ in the numerical ordering. It is obvious that as long as only components $x_{q+1}, \ldots, x_n$ are changed, the new vectors are also greater than $\mathbf{x}$ in the partial ordering. The last one of these vectors is $x_{q+1} = \ldots = x_n = 1$ with the rest of the variables unchanged. This shows that the calculation of $\mathbf{x}^*$, (2.13) is correct.

Proof 2°: In this case $\mathbf{x} \prec \tilde{\mathbf{x}}$ implies that $\mathbf{x} \leqq \tilde{\mathbf{x}}$, thus $\mathbf{x}^{*-} = (1, 1, \ldots, 1)$, which completes the proof.

Examples:

$\mathbf{x} = (0, 1, 0, 0, 1, 1, 0, 0)$
$\mathbf{x}^* = (0, 1, 0, 1, 0, 0, 0, 0)$
$\mathbf{x}^{*-} = (0, 1, 0, 0, 1, 1, 1, 1)$

$\mathbf{x} = (0, 1, 0, 0, 1, 1, 1, 1)$
$\mathbf{x}^* = (0, 1, 0, 1, 0, 0, 0, 0)$
$\mathbf{x}^{*-} = (0, 1, 0, 0, 1, 1, 1, 1)$

$\mathbf{x} = (0, 0, 0, 0, 0, 0, 0, 0)$
$\mathbf{x}^{*-} = (1, 1, 1, 1, 1, 1, 1, 1)$

$\mathbf{x} = (1, 1, 1, 0, 0, 0, 0, 0)$
$\mathbf{x}^{*-} = (1, 1, 1, 1, 1, 1, 1, 1).$

This calculation can easily be adapted for computers. The vector $\mathbf{x} \in D^n$ may be represented in one word of the computer.† Each component is placed to a bit of this word. This way we obtain exactly the binary representation of $s(\mathbf{x})$ defined by expression (2.7). It is easy to see that the calculation of $\mathbf{x}^*$ given by Lemma 2.2 is equivalent to the following.

Let us first determine $\mathbf{x}^-$ by $s(\mathbf{x}^-) = s(\mathbf{x}) - 1$. Then

$$\mathbf{x}^{*-} = \mathbf{x} \vee \mathbf{x}^-,$$

where $\vee$ is the logical disjunction taken componentwise (1 = true, 0 = false), which can be done by a single computer statement of bitwise disjunction on the numbers $s(\mathbf{x})$ and $s(\mathbf{x}) - 1$. (The computer uses binary representation.)

Examples:	$s(x)$	00101110	00000000	11000000
	$s(x^-)$	00101101	−11111111††	10111111
	$s(x^{*-})$	00101111	11111111	11111111
	$s(x^*)$	00110000.		

2.3 Summary of the Lawler–Bell algorithm and some remarks

The algorithm consists of the following steps.

Step 1: If the vector $\mathbf{x} = \mathbf{0}$ satisfies the constraints (2.10), then it is also the optimal solution of the problem. Then $\hat{g} = 0$, $\hat{\mathbf{x}} = 0$ and go to Step 8. Otherwise let $\hat{g} = U > g_0(x)$, $x \in D^n$.

Step 2: Vector $\mathbf{x}$ is replaced by the one immediately following it in the numerical ordering. (In binary representation 1 is added to $s(\mathbf{x})$.)

Step 3: Calculate $g_0(\mathbf{x})$.

Step 4: If $g_0(\mathbf{x}) \geqq \hat{g}$, then go to Step 7., otherwise to the next one.

Step 5: If $\mathbf{x}$ satisfies the constraints (2.10), substitute $\hat{\mathbf{x}}$ by $\mathbf{x}$ and $\hat{g}$ by $g(\mathbf{x})$, and go to Step 7, otherwise to the next one.

Step 6: If at least one of the inequalities

$$g_{i1}(\mathbf{x}^{*-}) - g_{i2}(\mathbf{x}) < 0 \quad (i = 1, \ldots, m)$$

is satisfied, then go to Step 7. In the opposite case, if $\mathbf{x} = (1, 1, \ldots, 1)$ go to Step 8, otherwise to Step 2.

Step 7: If $\mathbf{x}^*$ does not exist for the present $\mathbf{x}$, go to Step 8. Otherwise substitute $\mathbf{x}$ by $\mathbf{x}^*$ and go to Step 3.

Step 8: End of the algorithm. If $\hat{g} = U$, then no solution exists. Otherwise $\hat{\mathbf{x}}$ is the optimal solution and $\hat{g}$ is the corresponding objective function value.

Note 1: The non 0–1 discrete problems can be solved by the above method either substituting the variables by 0–1 variables or elaborating a version of the algorithm

† Supposing that n is not greater than the number of bits in one computer word. In the opposite case, several words are used in a similar way.

†† The computer stores negative numbers in complementer form.

for non 0–1 variables. In the latter case, from the theoretical point of view only the calculation of $\mathbf{x}^*$ and Step 2 have to be changed. Let us denote the upper bounds of the variables by $u_1, u_2, \ldots, u_n$, and suppose that all lower bounds are 0. Then subscripts $p < q$ are to be found with the following two properties:

(i) $x_p < u_p, \quad x_q > 0$

(ii) No pair (r, s) exists, such that $p < r < s$ and $x_r < u_r, \quad x_s > 0$

To put it into words, this means that we have to find the last unsaturated variable $(x_p < u_p)$ that is followed by a nonzero variable. If such subscripts p, q are found, then we can define $\mathbf{x}^*$ as

$$\begin{aligned} x_j^* &= x_j && \text{for all} \quad j < p \\ x_p^* &= x_p + 1 \\ x_j^* &= 0 && \text{for all} \quad j > p\,. \end{aligned}$$

Then $\mathbf{x}^{*-}$ is the immediate predecessor of $\mathbf{x}^*$.
Otherwise,(i.e. if no such pair p, q exists), then as before

$$\mathbf{x}^{*-} = (u_1, u_2, \ldots, u_n)$$

and no $\mathbf{x}^*$ exists.

To calculate the vector immediately following vector $\mathbf{x}$ in the numerical ordering, let

$$x_r < u_r, \quad x_{r+1} = u_{r+1}, \ldots, x_n = u_n$$

where $1 \leqq r \leqq n$. Then the desired vector is obviously the vector $\tilde{\mathbf{x}}$:

$$\begin{aligned} \tilde{x}_j &= x_j && j \leqq r - 1 \\ \tilde{x}_r &= x_r + 1 \\ \tilde{x}_j &= 0 && j \geqq r + 1\,. \end{aligned}$$

The above-described algorithm can be used without change, only the binary representation does not work.

Note 2: All linear, bounded discrete variable problems can be put into the form (2.9)–(2.11). First, the problem can be transformed to a 0–1 problem. Then the negative objective function coefficients will be reversed by the corresponding substitution $x_j' = 1 - x_j$. And the linear constraints are always differences of two nondecreasing functions.

Note 3: If in a nonlinear discrete problem the only difficulty is that the objective function is the difference of two nondecreasing functions but it is not a nondecreasing function itself, then a usual trick of introducing a new variable and a new constraint will help:

$$\begin{aligned} &\min z \\ &z - g_0(x) \geqq 0\,. \end{aligned}$$

Note 4: The experience of Lawler and Bell (1966) shows that problems up to twenty-five 0–1 variables may be effectively solved by this algorithm. For linear problems, especially of larger size, more efficient methods will be discussed later. It may be used, however, as a subprogram of a large adaptive system for solving discrete variable problems.

Note 5: The computer time used for solving different problems is largely dependent on the order of variables.

Example: Solve the following problem by the Lawler–Bell method.

$$\min 2x_1 + 3x_1x_4 + x_1x_5 + 3x_2x_3 + 10x_2x_5 + x_4x_5 + 5x_5 + 5x_6 = g_0(x)$$
$$g_1(\mathbf{x}) = 3x_1 + 5x_2 - 2x_3 + 3x_4 + 4x_5 - 2x_6 \geqq 6 \qquad (= b_1)$$
$$g_2(\mathbf{x}) = 6x_1 - 8x_2 + 7x_3 + 5x_5 \qquad \geqq 5 \qquad (= b_2)$$
$$g_3(\mathbf{x}) = -5x_1 + 10x_2 + 3x_3 + 3x_4 + 5x_5 - 5x_6 \geqq 7 \quad (= b_3)$$
$$x_j \in \{0, 1\} \quad (j = 1, \ldots, 6).$$

Using the notation introduced, the function $g_0(\mathbf{x})$ is already in appropriate form, furthermore

$$g_{11}(\mathbf{x}) = 3x_1 + 5x_2 + 3x_4 + 4x_5 \qquad g_{12}(\mathbf{x}) = 2x_3 + 2x_6$$
$$g_{21}(\mathbf{x}) = 6x_1 + 7x_3 + 5x_5 \qquad g_{22}(\mathbf{x}) = 8x_2$$
$$g_{31}(\mathbf{x}) = 10x_2 + 3x_3 + 3x_4 + 5x_5 \qquad g_{32}(\mathbf{x}) = 5x_1 + 5x_6 .$$

TABLE 2.1

	$\mathbf{x}$ 123456	$\mathbf{x}^{*-}$ 123456	$\hat{g}$	$g_0(\mathbf{x})$ if $\hat{g} < 100$	g_1 ($\geqq 6$)	g_2 ($\geqq 5$)	g_3 ($\geqq 7$)	$g_{i1}(\mathbf{x}^{*-}) - g_{i2}(\mathbf{x}) < b_i$ $i = 1, 2, 3$
1	000000	111111	100		0			
2	000001	000001			−2			
3	000010	000011			4			
4	000100	000111			3			$i = 1$ ⊗
5	000101	000101			1			
6	000110	000111	$\hat{g} = 6$	8	7	5	8	
7	001000	001111		0	−2			
8	010000	011111		0	5			$i = 1$ ⊗
9	100000	111111		2	3			$i = 2$ ⊗
10	100001	100001		7	1			
11	100010	100011		$8 \geqq \hat{g}$ ⊗				
12	100100	100111		5	6	6	−2	
13	101000	101111		2	1			$i = 3$ ⊗
14	110000	111111		2	8	−2		$i = 3$ ⊗
15	110001	110001		7	6	−2		
16	110010	110011		$22 > \hat{g}$ ⊗				
17	110100	110111		5	11	−2		
18	111000	111111	$\hat{g} = 5$	5	6	5	8	$i = 2$ ⊗

Table 2.1 shows the use of the algorithm to solve the problem. All columns are completed in their order until one of the conditions described in Lemma 2.1 is satisfied in which case the remaining conditions are not checked and the calculation is continued with $\mathbf{x}^*$. Otherwise the evaluation of $\mathbf{x}$ is continued by the examination of the solution immediately following $\mathbf{x}$ in the numerical ordering. (1 is added to $s(\mathbf{x})$ in the binary representation.) The sign $\otimes$ shows if somewhere a jumping condition (one of the conditions 1°–3° of Lemma 2.1) is satisfied. In the last column also the subscript of the constraint is written, which made the jump possible. If $\mathbf{x}$ and $\mathbf{x}^{*-}$ are identical (the last components of $\mathbf{x}$ are 01) then condition 3° of Lemma 2.1 is useless and therefore the checking is omitted because the jump from $\mathbf{x}$ to $\mathbf{x}^*$ is not a real jump, nothing is left out. The value of $\hat{g}$ is written in column 3 only where it changes.

The optimal solution has been obtained in Step 18. It should be noted that the optimal solution is not usually obtained in the last step, it was purely by chance that it happened in our example.

2.4 The concept of the ordered pseudo-solution

In connection with the Lawler–Bell algorithm it was mentioned that the order of variables is important. Although it would be possible to establish good rules for ordering the variables, it is obvious that this is of little practical value because after fixing a few variables the problem has a substantially different form, for which a different order of variables would be favourable. In other words a flexible enumeration algorithm is needed in which the next variable to be fixed may be chosen arbitrarily. In this way, the information gathered during the process of the algorithm may also be utilized in the further steps.

Let us consider an arbitrary set

$$S \subset D^n = \{\mathbf{x} \mid \mathbf{x} \in E^n, \quad x_j \in \{0, 1\} \quad (j = 1, 2, \ldots, n)\}$$

where E^n is the set of n-dimensional vectors. The set S is implicitly determined by equations, inequalities and logical conditions. In the next section we shall give a general framework for obtaining explicitly all elements of the set S.

The last section of Chapter 2 shows the application of the algorithm for optimization problems without determining all feasible solutions, i.e. all elements of the set S.

For this purpose Balas' pseudo-solution and its extensions will be introduced.

We shall give, however, the basic definitions in a substantially different form from that of Balas. One reason is to get closer to the standard notation and concepts of set theory, the other is to obtain a clearer view of solution sets and operations on them.

Definition 2.3: *The set defined by the following expression is called the pseudo-solution:*

$$Q = \{\mathbf{x} \mid \mathbf{x} \in D^n, x_{j_1} = \delta_{j_1}, \ldots, x_{j_k} = \delta_{j_k}\},$$

where

$$\{j_1, j_2, \ldots, j_k\} \subset \{1, 2, \ldots, n\}$$

is a given set of indices and $\delta_{j_1}, \ldots, \delta_{j_k}$ *are constants having one of the values* 0 or 1.

The dimension is usually evident and therefore the term $\mathbf{x} \in D^n$ is often omitted. The dimension n is always the same as that of the examined set S.

If pseudo-solution Q contains only one element (all the variables have been fixed), then we do not distinguish the set Q from its single element. Therefore it is meaningful to state in this case, for example, that $Q \in S$ instead of $Q \subset S$.

Example: Let

$$Q = \{\mathbf{x} \mid \mathbf{x} \in D^6, x_2 = 1, x_3 = 0, x_5 = 1\}$$

in the space E^6 and

$$S = \{\mathbf{x} \mid \mathbf{x} \in D^6, \sum_{j=1}^{6} x_j = 3\}.$$

Then the elements of set Q are the following

(0, 1, 0, 0, 1, 0)	(1, 1, 0, 0, 1, 0)
(0, 1, 0, 0, 1, 1)	(1, 1, 0, 0, 1, 1)
(0, 1, 0, 1, 1, 0)	(1, 1, 0, 1, 1, 0)
(0, 1, 0, 1, 1, 1)	(1, 1, 0, 1, 1, 1)

and

$$Q \cap S = \{(0, 1, 0, 0, 1, 1), \quad (0, 1, 0, 1, 1, 0), \quad (1, 1, 0, 0, 1, 0)\}.$$

The notation may be simplified by introducing the terms

$$i \quad \text{instead of} \quad x_i = 1$$
$$\bar{i} \quad \text{instead of} \quad x_i = 0.$$

Example: $Q = \{\mathbf{x} \mid x_2 = 1, x_3 = 0, x_5 = 1\} = \{x \mid 2, \bar{3}, 5\}.$

It follows from the definition that the order of the terms has no importance in the pseudo-solution. The variables having a value in the pseudo-solution are called fixed, the others are called free variables. If there is no fixed variable in a pseudo-solution, then it is the set of all solutions, i.e. all elements of D^n. If there are n fixed variables in a pseudo-solution of n dimensions, then this is a real solution. If it is also an element of the examined set S, then it is called a feasible solution.

Note: The concept of the pseudo-solution is important because if a discrete optimization problem is restricted to a pseudo-solution, then a new problem is obtained which is of the same type as the original one. Namely, only the values of certain variables are fixed. Other types of solution sets do not have this very useful property.

The following part of the present chapter may be omitted at first reading because it contains the precise mathematical concept of the ordered pseudo-solution and related functions. After this part a suggestive explanation of this notion will make the further parts understandable. Later on, a return to this precise mathematical description may be useful from several points of view. It is now suggested that reading be continued at the "Summary of the ordered pseudo-solution concept".

To enable the concept of the pseudo-solution to be more fully understood and to make it more useful, we introduce the following definition:

Definition: 2.4: *The* **generator** *of the pseudo-solution*

$$Q = \{\mathbf{x} \mid \mathbf{x} \in D^n,\ x_{i_1} = \delta_{i_1}, \ldots,\ x_{i_k} = \delta_{i_k}\}$$

is an ordered pair

$$G = G(Q) = (J, B),$$

where

$$J = \{j_1, j_2, \ldots, j_k\}$$

and B is a binary function on the set $\{1, 2, \ldots, n\}$ furthermore

$$B(j_r) = \delta_{i_r} \qquad (r = 1, 2, \ldots, k)$$

$$B(j) = 0 \qquad j \in \{1, 2, \ldots, n\} - J\,.$$

It is easily seen that a pseudo-solution and its generator uniquely determine each other.

Unfortunately the conjunction or disjunction of two pseudo-solutions are, in general, no longer pseudo-solutions. In order to define substitutes for these notions, let us consider two arbitrary pseudo-solutions with the common fixed variables $x_{j_r}, \ldots, x_{j_c}$:

$$Q_1 = \{\mathbf{x} \mid \mathbf{x}D^n,\ x_{j_1} = \delta_{i_1}, \ldots, x_{j_r} = \delta_{i_r}, \ldots, x_{j_s} = \delta_{i_s}\}$$
$$Q_2 = \{\mathbf{x} \mid \mathbf{x} \in D^n,\ x_{j_r} = \delta'_{i_r}, \ldots, x_{j_s} = \delta'_{i_s}, \ldots, x_{j_t} = \delta'_{i_t}\}$$

and their generators

$$G_1 = (J_1, B_1), \qquad G_2 = (J_2, B_2)$$

where

$$J_1 = \{j_1, \ldots, j_r, \ldots, j_s\} \qquad J_2 = \{j_r, \ldots, j_s, \ldots, j_t\}$$
$$B_1(j) = \delta_j \qquad (j \in J_1)$$
$$B_2(j) = S_t \qquad (j \in J_2).$$

Let us define the following set of subscripts

$$\tilde{J} = \{j \mid j \in J_1 \cap J_2, \qquad B_1(j) \neq B_2(j)\}\,,$$

and the binary functions

$$\hat{B}(j) = B_1(j)B_2(j)$$

$$\tilde{B}(j) = \begin{cases} B_1(j)B_2(j) & \text{if} \quad j \in J_1 \cap J_2 \\ B_1(j) & \text{if} \quad j \notin J_2 \\ B_2(j) & \text{if} \quad j \notin J_1\,. \end{cases}$$

Then we can introduce the conjunction, generalized conjunction, disjunction, and generalized disjunction, respectively, of two generators:

$$G_1 \cap G_2 = (J_1 \cup J_2, \hat{B}) \qquad \text{if} \quad \tilde{J} = \emptyset$$
$$G_1 \Delta G_2 = (J_1 \cap J_2 - \tilde{J}, \hat{B})$$
$$G_1 \cup G_2 = (J_1 \cup J_2, \tilde{B}) \qquad \text{if} \quad \tilde{J} = \emptyset$$
$$G_1 \nabla G_2 = (J_1 \cup J_2 - \tilde{J}, \tilde{B}).$$

Let us denote by $Q(G)$ the pseudo-solution corresponding to the generator G.

Example 1:

$$Q(G_1) = \{\mathbf{x} \mid 1, \bar{2}, 3, \bar{4}\}$$
$$Q(G_2) = \{\mathbf{x} \mid 3, \bar{4}, \bar{5}, 6\}$$
$$Q(G_1 \Delta G_2) = Q(G_1 \cap G_2) = \{\mathbf{x} \mid 3, \bar{4}\}$$
$$Q(G_1 \nabla G_2) = Q(G_1 \cup G_2) = \{\mathbf{x} \mid 1, \bar{2}, 3, \bar{4}, \bar{5}, 6\}$$

Example 2:

$$Q(G_3) = \{\mathbf{x} \mid 1, 3, \bar{4}\}$$
$$Q(G_4) = \{\mathbf{x} \mid 3, 4, 5, \bar{6}\}$$
$$Q(G_3 \Delta G_4) = \{\mathbf{x} \mid 3\}$$
$$Q(G_3 \nabla G_4) = \{\mathbf{x} \mid 1, 3, 5, \bar{6}\}$$

Lemma 2.3:

1° $Q(G_1 \cap G_2) \supset Q(G_1) \cup Q(G_2)$, if $\tilde{J} = \emptyset$

2° $Q(G_1 \Delta G_2) \supset Q(\tilde{G}_1) \cup Q(\tilde{G}_2)$,

3° $Q(G_1 \cup G_2) = Q(G_1) \cap Q(G_2)$, if $\tilde{J} = \emptyset$

4° $Q(G_1 \nabla G_2) = Q(\tilde{G}_1) \cap Q(\tilde{G}_2)$

where

$$\tilde{G}_1 = (J_1 - \tilde{J}, B_1) \quad \text{and} \quad \tilde{G}_2 = (J_2 - \tilde{J}, B_2).$$

Proof: Relations 1° and 3° are special cases of relations 2° and 4° respectively, but the former ones show the symmetricity of relations better.

2° As the relation is symmetric in G_1 and G_2, let us consider an arbitrary element $\hat{\mathbf{x}} \in Q(G_1)$, that is,

$$\hat{x}_{j_1} = \delta_{j_1}, \ldots, \hat{x}_{j_s} = \delta_{j_s}, \ldots, \hat{x}_{j_r} = \delta_{j_r}.$$

According to the definition of function $\hat{B}(j)$

$$\hat{B}(j) = B_1(j) \quad \text{for all} \quad j \in J_1 \cap J_2 - \tilde{J} \subset J_1 - \tilde{J}_1,$$

that is, all the components fixed in pseudo-solution $Q(G_1 \Delta G_2)$ take the required values for $\hat{\mathbf{x}} \in Q(G_1 \Delta G_2)$.

4° Let us suppose that an arbitrary element $\hat{\mathbf{x}} \in Q(\tilde{G}_1) \cap Q(\tilde{G}_2)$. According to the definition of function $\tilde{B}(j)$

$$\tilde{B}(j) = B_1(j) \quad j \in J_1 - \tilde{J}$$
$$\tilde{B}(j) = B_2(j) \quad j \in J_2 - \tilde{J}$$

that is, variables x_j take the right value for

$$j \in (J_1 - \tilde{J}) \cup (J_2 - \tilde{J}) = J_1 \cup J_2 - \tilde{J}$$

and thus $\hat{x} \in Q(G_1 \nabla G_2)$

This proof can also be followed backwards, i.e. the relation

$$Q(G_1 \nabla G_2) \subset Q(\tilde{G}_1) \cap Q(\tilde{G}_2)$$

also holds, which completes the proof.

Note 1: It can easily be shown that the left hand sides of relations 1°–4° are, in general, the smallest pseudo-solutions satisfying these relations, i.e. any further variable fixing makes the relations invalid.

Note 2: Similar relations hold true for any number of pseudo-solutions.

Note 3: These relations may be used together with relaxation procedures or simply when a solution set is already decomposed into many pseudo-solutions.

Note 4: The relation 2° remains true if the $\sim$ is deleted from the right hand side as $Q(\tilde{G}_1) \supset \supset Q(G_1)$ and $Q(\tilde{G}_2) \supset Q(G_2)$.

We can express also the difference of two pseudo-solutions as the disjunction of pseudo-solutions. Using the notation Q_1, Q_2 and $\tilde{J}$ introduced above:

$$Q_1 - Q_2 = Q_1 \quad \text{if} \quad \tilde{J} \neq \emptyset$$

and

$$Q_1 - Q_2 = \bigcup_{i=1}^{t-s} Q_i \quad \text{if} \quad \tilde{J} = \emptyset \tag{2.14}$$

where

$$Q_i = \{\mathbf{x} \mid \mathbf{x} \in D^n,\ x_{l_1} = \delta_{l_1}, \ldots, x_{l_r} = \delta_{i_r},\ x_{l_{s+i}} = 1 - \delta'_{l_{s+i}}\}\ .$$

In what follows we shall need a refining partial ordering among the pseudo-solutions formed by the algorithm. For this purpose we introduce

Definition 2.5: *The ordered triple*

$$\phi = (J, B, P) \tag{2.15}$$

is called ordered pseudo-solution, where

$$(J, B) = G \tag{2.16}$$

is the generator of a pseudo-solution, (see Definition 2.4) *and P is a mapping of the set*

$J = \{j_1, j_2, \ldots, j_k\}$ *into the set* $I_k = \{1, 2, \ldots, k\}$, that is,

$$\{P(j_1), P(j_2), \ldots, P(j_k)\} = \{1, 2, \ldots, k\}\ . \tag{2.17}$$

The unordered pseudo-solution corresponding to the ordered pseudo-solution, defined above, is obviously

$$U(\phi) = Q(G),$$

that is, the pseudo-solution the generator of which is (J, B). The ordered pseudo-solution will often be denoted in a more expressive way

$$(x_{j_1} = \delta_{j_1}, x_{j_2} = \delta_{j_2}, \ldots, x_{j_k} = \delta_{j_k}), \tag{2.18}$$

from which the defining triple (2.15) may easily be determined:

$$\begin{aligned} &J = \{j_1, j_2, \ldots, j_k\} \\ &B(j_r) = \delta_{j_r} \qquad (r = 1, 2, \ldots, k) \\ &P(j_r) = r \qquad (r = 1, 2, \ldots, k). \end{aligned} \tag{2.19}$$

Example: $\phi_1 = (x_2 = 1, x_1 = 0, x_5 = 1)$
$\phi_2 = (x_5 = 1, x_2 = 1, x_1 = 0)$,

then obviously $\phi_1 \neq \phi_2$, but $U(\phi_1) = U(\phi_2)$ because considering the corresponding triples

$$J_1 = J_2, B_1 = B_2, P_1 \neq P_2 .$$

If an ordered pseudo-solution is given in the form of (2.15) then form (2.18) may also be obtained because J is the set of subscripts of fixed variables, function B determines the fixing values, and the mapping P provides the order of fixing terms.

A short notation may be used even without parentheses (only for the ordered pseudo-solution!).

Example: $\phi = (x_3 = 1, x_5 = 0, x_2 = 0) = (3, \bar{5}, \bar{2}) = 3\,\bar{5}\,\bar{2}$.

Sometimes, because we want to prevent the forming of certain ordered pseudo-solutions as the enumeration algorithm proceeds, we introduce the notion of tied variables.

Now let us introduce an additional binary function $T(j)$ $(j \in J)$ which may take only the values 0 or 1. It is called a tying function assigned to the ordered pseudo-solution

$$\phi = (x_{j_1} = \delta_{j_1}, x_{j_2} = \delta_{j_2}, \ldots, x_{j_k} = \delta_{j_k})$$

where J is the set of fixed variables

$$J = \{j_1, j_2, \ldots, j_k\} .$$

If

$$T(j_r) = 1 ,$$

then variable x_{j_r} is called a tied variable. This means that ordered pseudo-solutions of the form

$$\phi' = (x_{j_1} = \delta_{j_1}, \ldots, x_{j_{r-1}} = \delta_{j_{r-1}}, x_{j_r} = 1 - \delta_{j_r}, \ldots)$$

are excluded from the algorithm.

This tying function and the tying of variables in general, will have an important role in the enumeration of algorithms. (See the examples and the explanation below.)

In hand calculations the tying of a variable will be denoted by underlining the corresponding variable fixing.

Summary of the concept of an ordered pseudo-solution

Let us consider a pseudo-solution

$$Q = \{\mathbf{x} \mid \mathbf{x}_{j_1} = \delta_{j_1}, x_{j_2} = \delta_{j_2}, \ldots, x_{j_k} = \delta_{j_k}\} .$$

If we wish to emphasize the order of fixing of variables, then we use the so-called ordered pseudo-solution

$$\phi = (x_{j_1} = \delta_{j_1}, x_{j_2} = \delta_{j_2}, \ldots, x_{j_k} = \delta_{j_k}),$$

which means that the above pseudo-solution Q is used with the additional information that the variables have been fixed in the order $x_{j_1}, x_{j_2}, \ldots x_{j_k}$. In other words, an ordered pseudo-solution consists of a pseudo-solution plus the order of fixing terms. If we wish to disregard the ordering, i.e. to obtain the solution set of an ordered pseudo-solution, then we use the letter U ("unordering"). In the above case

$$U(\phi) = Q\,.$$

Example: Obviously the ordered pseudo-solutions

$$\phi_1 = (x_1 = 0, x_3 = 1, x_7 = 1)$$
$$\phi_2 = (x_7 = 1, x_1 = 0, x_3 = 1)$$

are different because the orders of variable fixings are not the same even though the corresponding solution sets are identical, that is,

$$\phi_1 \neq \phi_2, \quad \text{but} \quad U(\phi_1) = U(\phi_2).$$

The concept of an ordered pseudo-solution has been introduced in order to construct a framework for discrete programming algorithms especially of enumeration type. An ordered pseudo-solution describes a solution set (the corresponding pseudo-solution) and carries the state of the enumeration as an additional item of information. For this purpose, however, another notion should also be introduced. A variable fixing term $x_{j_r} = \delta_{j_r}$ in an ordered pseudo-solution

$$\phi = (x_{j_1} = \delta_{j_1}, \ldots, x_{j_{r-1}} = \delta_{j_{r-1}}, x_{j_r} = \delta_{j_r}, \ldots, x_{j_k} = \delta_{j_k})$$

is called tied if the fixing $x_{j_r} = \delta_{j_r}$ must not be deleted or altered until at least one of the preceding fixings $x_{j_1} = \delta_{j_1}, \ldots, x_{j_{r-1}} = \delta_{j_{r-1}}$ is deleted or altered. The variable x_{j_r} is also called a tied variable in the pseudo-solution ϕ. In our short notation the tying will be denoted by underlining the fixing term.

Example 1: $\phi = (x_4 = 0, x_3 = 1, \underline{x_5 = 0})$.

This tying indicates that if $x_4 = 0$ and $x_3 = 1$, then variable x_5 must take on the value 0.

Example 2: $\phi = (\underline{x_3 = 1}, x_2 = 0, x_7 = 1, \underline{x_9 = 0}, \underline{x_3 = 1})$.

These tyings show that x_3 must take on the value 1 unconditionally, furthermore if $x_2 = 0$ and $x_7 = 1$, then variables x_9 and x_3 must take on the values 0 and 1, respectively.

There may be two different reasons for tying a variable. One of them, as we have mentioned, is that the opposite value of the variable cannot result in a feasible solution (a point of the set S); the other is that the opposite value has already been examined under the restrictions of the preceding terms. Examining the relations of ordered pseudo-solutions we introduce:

Definition 2.6: *An ordered pseudo-solution ϕ_2 is called a descendant of pseudo-solution ϕ_1 if ϕ_2 can be obtained from ϕ_1 simply by adding new fixing terms to it. Then ϕ_1 is called the predecessor of ϕ_2. If only one term is added then they are called* **immediate** *descendant and predecessor, respectively.*

Definition 2.7: *ϕ_2, a descendant of the ordered pseudo-solution ϕ_1 is called a consequence of ϕ_1 related to the set S, if*

$$U(\phi_1) \cap S = U(\phi_2) \cap S . \tag{2.20}$$

If ϕ_2 is an immediate descendant of ϕ_1 and relation (2.20) *holds, then ϕ_2 is called the* **immediate consequence** *of ϕ_1 related to the set S.*

If it is not misleading, the words "related to the set S" will be omitted.

Example: Let

$$S = \{\mathbf{x} \mid \mathbf{x} \in D^8, \quad \sum_{j=1}^{3} x_j \leqq 1, \quad \sum_{j=6}^{8} x_j \geqq 2\}$$

$$\phi_1 = (\bar{8}, 3),$$

then obviously

$$\phi_2 = (\bar{8}, 3, 6, 7, \bar{1}, \bar{2})$$

is a consequence of ϕ_1, thus the form

$$\phi_2 = (\bar{8}, 3, \underline{6}, \underline{7}, \underline{\bar{1}}, \underline{\bar{2}})$$

is completely correct.

In order to simplify the notation, the terms of the pseudo-solutions will often be denoted by greek letters, that is,

$$Q_1 = \{x \mid x_{j_1} = \delta_{j_1}, \ldots, x_{j_k} = \delta_{j_k}\}$$

is substituted by

$$Q_1 = \{x \mid \alpha_1, \ldots, \alpha_k\} .$$

Lines under the terms are used to tie the term as before, and lines over the term for changing the term to the opposite variable fixing.

Examples: α_1: $x_5 = 0$, $\quad \underline{\bar{\alpha}_1}$: $\underline{x_5 = 1}$

α_2: $\bar{3}$ $\quad \bar{\alpha}_2$: 3

2.5 Skeleton of enumeration algorithms

A general algorithm will now be given which includes most of the enumeration algorithms if the extensions of Chapter 3 are also considered. The algorithm is rather a skeleton, which may be "dressed" in many different ways and thus many known and newly developed algorithms may be obtained. The algorithm is based

on the backtracking procedure of Walker (1960), and the papers of Balas (1965), and of Glover (1965). It will be presented first for the case of 0–1 pure integer programming and for single branch enumeration algorithms. In a later chapter it will be shown how it may be extended for general integer programming problems and how the multi-branch algorithms may be included in this framework.

For the sake of simplicity, first of all we shall determine explicitly all elements of an implicitly given discrete set $S \subset D^n$. S is determined by equations, inequalities, logical conditions, etc. It should be noted, however, that in optimization problems the determination of all feasible solutions will by no means be necessary – as will be illustrated later in the present chapter.

The pseudo-solution $\{\mathbf{x} \mid \mathbf{x} \in D^n\}$ contains no fixed variable, therefore it may also be considered as an ordered pseudo-solution, $(\emptyset)$, because no order should be imposed on the set $J = \emptyset$.

Let us consider the set X determined by the following algorithm (it is always continued at the next step unless otherwise noted):

Step 1: $\phi = (\emptyset)$; $X = \emptyset$.

Step 2: The present ordered pseudo-solution ϕ is substituted by an immediate consequence of ϕ related to the set S, tying (underlining) the last, newly added term. The result is denoted by ϕ and the process is repeated as many times as possible. If no more consequences can be found:

Step 3: If it can be shown that $U(\phi) \cap S = \emptyset$, then go to Step 5, otherwise:

Step 4: (a) If there are free variables in the present ordered pseudo-solution ϕ, then one of them is fixed (but not tied) at value 0 or 1 and added to ϕ as a last term. The result is again denoted by ϕ. Go to Step 2.

(b) If there is no free variable in ϕ, then if $U(\phi) \subset S$, then $X := X \cup U(\phi)$, that is the new feasible solution is registered.† In all cases continue at the next step.

Step 5: If ϕ contains no untied variables, the algorithm is **terminated.** Otherwise go to Step 6.

Step 6: (Backtracking) The last untied variable in the present ordered pseudo-solution is changed to its opposite value (from 0 to 1 or from 1 to 0) and tied. All the following terms are omitted. Go to Step 2.

It will be proven in Theorem 2.1 that at the end of the algorithm all elements of S are explicitly determined, that is $S = X$.

Note ad 2: If some of the consequences cannot be obtained by the applied criteria (tests) the algorithm works correctly, but more steps are necessary.

Note ad 4a: The choice of free variable and its value is of basic importance, especially at the beginning. Therefore a measure of "goodness" of variables is used to accelerate the method (see Chapters 5, 6, 9, and 11).

† If all variables are fixed in ϕ, then $U(\phi)$ consists of only one solution. In this case the set $U(\phi)$ is considered identical with this single element.

Note ad 4b: Using good criteria relatively few unnecessary trials of type $U(\phi) \subset S$ is obtained. If the problem is

$$\min_{x \in S} f(x),$$

and function $f(x)$ may take only integer values, then after a new feasible solution $x^k \in S$ has been found, the so-called objective function constraint is corrected:

$$f(x) \leqq f(x^k) - 1 .$$

Thus we are interested only in the feasible solutions better than the best one found so far.

Theorem 2.1: *The above algorithm comes to an end after finite number of steps* i*without repetition of any pseudo-solutions and then* $S = X$.

Proof: First of all we shall show that no pseudo-solution can occur twice throughout the algorithm, not even in the form of two different ordered pseudo-solutions.

For this purpose, let us take an arbitrary positive integer N and consider two different pseudo-solutions

$$Q_1 = \{\mathbf{x} \mid \beta_1, \beta_2, \ldots, \beta_p\},$$
$$Q_2 = \{\mathbf{x} \mid \gamma_1, \gamma_2, \ldots, \gamma_q\},$$

which occurred in the first N steps of the algorithm in the form of ordered pseudo-solutions and let $p \geqq q$. Some of the terms $\beta_1, \ldots, \beta_p$ may be identical with $\gamma_1, \ldots, \gamma_q$, but not all of them. In this case it can be uniquely determined which one of Q_1 and Q_2 appeared first. This can be done in the following way:

Let us denote the subscript of the variable fixed first by j_1. Then $j_1 \in J_1$ and $j_1 \in J_2$ (where J_1 and J_2 are the sets of subscripts of fixed variables in Q_1 and Q_2 respectively) because the omission of a term (carried out only in Step 6) is possible only if a previous fixed variable has been changed and j_1 is the first fixed variable. If $\delta^1_{j_1} = 1 - \delta'^2_{j_1}$ (where $\delta^1_{j_1}$ and $\delta'^2_{j_1}$, are the values of variable x_{j_1} in Q_1 and Q_2 respectively) then the order is obvious according to whether $x_{j_1} = 1$ or $x_{j_1} = 0$ came first in the algorithm. If both Q_1 and Q_2 contain the same defining term $\alpha_1 : x_{j_1} = \delta_{j_1}$, then only the pseudo-solutions containing the term α_1 should be considered because after the first pseudo-solution with α_1 is generated, no more pseudo-solutions with α_1 can be obtained according to the backtracking rule. In the branch defined by α_1, j_2, the subscript of the variable fixed second plays the role of j_1 and the reasoning can be repeated word for word. When continuing the process (as Q_1 and Q_2 are not identical), the following cases may occur (supposing that the terms $\alpha_1, \alpha_2, \ldots, \alpha_r$ are identical in Q_1 and Q_2 but the next fixing term generated in this branch, α_{r+1}, is not in both Q_1 and Q_2):

(i) Q_1 contains α_{r+1} and Q_2 contains $\bar{\alpha}_{r+1}$

(ii) Q_2 contains α_{r+1} and Q_1 contains $\bar{\alpha}_{r+1}$

(iii) Q_1 contains α_{r+1} and Q_2 contains neither α_{r+1} nor $\bar{\alpha}_{r+1}$

(iv) Q_2 contains α_{r+1} and Q_1 contains neither α_{r+1} nor $\bar{\alpha}_{r+1}$.

Cases Q_1 (iii) and (Q_2 (iv) may occur only if Q_1 and Q_2 respectively, contain no terms other than $\alpha_1, \ldots, \alpha_r$.

Obviously both Q_1 and Q_2 must occur in the branch defined by $\alpha_1, \alpha_2, \ldots, \alpha_r$ — and only here (see the α_1 branch above). Thus the next term determines the order uniquely. In cases (i) and (iv), Q_1 precedes Q_2; in cases (ii) and (iii), the opposite ordering occurs. This implies that if there was a pseudo-solution Q_1 in the form of two different ordered pseudo-solution ϕ_1 and ϕ_3, then between them there is no other pseudo-solution Q_2 in any ordered form. But ϕ_1 and ϕ_3 cannot follow each other directly as a consequence of the rules for forming ordered pseudo-solutions one after the other. At each step either a new term is added or one of the variables changes its value. The above reasoning is true for any positive integer N, therefore no repetition of ordered pseudo-solutions is possible, hence the algorithm is always finite.

It remains to show that at the end of the algorithm all elements of S are found, that is, $S = X$. $X \subset S$ is obvious because of Step 4(b). To show, that $X \supset S$ let us suppose indirectly that there is a vector $\hat{\mathbf{x}} \in S$ and $\hat{\mathbf{x}} \notin X$. Let us denote by $\hat{\phi}$ the ordered pseudo-solution generated by the algorithm with the following properties

1° $\hat{\mathbf{x}} \in U(\hat{\phi})$

2° No ϕ' is generated, such that

$$\hat{\mathbf{x}} \in U(\phi') \subset U(\hat{\phi}).$$

Such a $\hat{\phi}$ certainly exists because $\hat{\mathbf{x}} \in U(\phi_0) = D^n$. If the algorithm was terminated at $\hat{\phi}$, then all of its terms are underlined (all the variables are tied) and we are at Step 5. Then we came from Step 3, where it turned out that

$$U(\phi) \cap S = \emptyset$$

which is false.

If the algorithm was not terminated at $\hat{\phi}$, then variable x_{j_k}, fixed next in the algorithm, only took the opposite value $x_{j_k} = 1 - \hat{x}_{j_k}$ because of the property 2°, that is, x_{j_k} is tied at this value as a consequence (which is again false — see Definition 2.6), of

$$\hat{\mathbf{x}} \in U(\hat{\phi}) \cap S - \{\mathbf{x} \mid \mathbf{x} \in U(\hat{\phi}), \quad x_{j_k} = 1 - \hat{x}_{j_k}\} \cap S \neq \emptyset$$

and this completes the proof.

2.6 Application of enumeration algorithms to pure 0–1 problems

Now we shall solve a few problems on the basis of Section 2.5 using *ad hoc* tests. Systematic rules and tests, i.e. complete algorithms, will be presented in the subsequent chapters.

Example 1: As a first exercise, just to see how the backtracking procedure works, let $S_1 = D^3$, all vectors of 0–1 components in the three-dimensional Euclidean space. Let us give all elements of S_1 by the procedure described in Section 2.5. Now we have to give a rule for the free choice of Step 4(a). For simplicity we shall always choose the free variable of lowest subscript and assign the value 1 at first. (If we had had an objective function, then we might have used different rules as in later examples.)

Now the algorithm provides the steps given in Table 2.2.

Example 2: Let us determine explicitly the set $S_2 \subset D^6$ defined by the following constraints

$$x_1 + x_2 + x_3 \geqq x_4 + x_5 + x_6 \tag{2.21}$$

$$x_j = x_{j+1} = 1 \Rightarrow x_{j+2} = 0 \quad (j = 1, \ldots, 4) \tag{2.22}$$

$$x_j = x_{j+1} = 0 \Rightarrow x_{j+2} = 1 \quad (j = 1, \ldots, 4) \tag{2.23}$$

$$x_j \in \{0, 1\} \quad (j = 1, \ldots, 6) \tag{2.24}$$

TABLE 2.2

Pseudo-solution			Solution x_1	x_2	x_3
$\emptyset$					
1					
1	2				
1	2	3	1	1	1
1	2	$\bar{3}$	1	1	0
1	$\bar{2}$				
1	$\bar{2}$	3	1	0	1
1	$\bar{2}$	$\bar{3}$	1	0	0
$\bar{1}$					
$\bar{1}$	2				
$\bar{1}$	2	3	0	1	1
$\bar{1}$	2	$\bar{3}$	0	1	0
$\bar{1}$	$\bar{2}$				
$\bar{1}$	$\bar{2}$	3	0	0	1
$\bar{1}$	$\bar{2}$	$\bar{3}$	0	0	0

where the sign $\Rightarrow$ means "implies". The meaning of constraints (2.22) and (2.23) is, that no consecutive three variables should take identical values. This could have been expressed in algebraic form, i.e. in the space D^6 the set of constraints (2.22) is equivalent to following ones:

$$x_j + x_{j+1} + x_{j+2} \leqq 2 \quad (j = 1, \ldots, 4).$$

It will be clear however, that forms (2.22) and (2.23) will be more suitable for our approach than the algebraic formulation. The first constraint can also be expressed in logical form. This means no restriction for the first three variables if they are chosen first. In which case as soon as the sum of fixed variables in the second three equals the sum of the first three variables, the remaining variables must take the value 0 (they are not only fixed but also tied). In such a way, no point of the set $D^6 - S_2$ will ever be generated. Table 2.3 contains the steps of the algorithm. If the new pseudo-solution is obtained from the old one by adding a new term (a new fixed variable), then there is no need to start a new row. The second column

TABLE 2.3

No.	Pseudo-solution	Feasible solutions (points of set S_2)						Tying of variables other than backtracking (constraint number)
		x_1	x_2	x_3	x_4	x_5	x_6	
1	$12\bar{3}45\bar{6}$	1	1	0	1	1	0	$\bar{3}$: (22); $\bar{6}$: (22), (21)
2	$\bar{5}6$	1	1	0	1	0	1	
3	$\bar{6}$	1	1	0	1	0	0	
4	$\bar{4}56$	1	1	0	0	1	1	5 : (23)
5	$\bar{6}$	1	1	0	0	1	0	
6	$\bar{2}34\bar{5}6$	1	0	1	1	0	1	$\bar{5}$: (22)
7	$\bar{6}$	1	0	1	0	1	1	
8	$\bar{4}56$	1	0	1	0	1	1	
9	$\bar{6}$	1	0	1	0	1	0	
10	$\bar{5}6$	1	0	1	0	1	1	6 : (23)
11	$\bar{3}45\bar{6}$	1	0	0	1	0	0	4 : (23); $\bar{5}$: (21); $\bar{6}$: (21)
12	$\bar{1}23\bar{4}56$	0	1	1	0	1	1	$\bar{4}$: (22)
13	$\bar{6}$	0	1	1	0	1	0	
14	$\bar{5}6$	0	1	1	0	0	1	6 : (23)
15	$\bar{3}45\bar{6}$	0	1	0	1	0	0	$\bar{5}$: (21); $\bar{6}$: (21)
16	$\bar{4}5\bar{6}$	0	1	0	0	1	0	5 : (23); $\bar{6}$: (21)
17	$\bar{2}34\bar{5}\bar{6}$	0	0	1	1	0	0	3 : (23); $\bar{5}$: (21), (23); $\bar{6}$: (21)
18	$\bar{4}5\bar{6}$	0	0	1	0	1	0	$\bar{6}$: (21)
19	$\bar{1}23\bar{4}5\bar{6}$	0	0	1	0	0	1	6 : (23)

contains the elements of set S_2. The third column explains the tying of variables, if any, by giving the number of constraint set which made it necessary. As in the first example, in the case of free choice it is always the next free variable that is fixed at 1. For the sake of better arrangement in each row, the terms before the changing term are not written; they are identical to those of the previous row.

Example 3: The solution of this example will already show the real merit of the enumeration algorithm. However, the rules are still *ad hoc* tests and special choices of the variables to be fixed next. Generally applicable enumeration algorithms will be discussed in detail in further chapters. The problem is the following.

Minimize

$$z = \sum_{j=1}^{8} x_j \tag{2.25}$$

on the domain S defined by the following constraints

$$x_1(x_1 + x_2 + 5x_3 + x_4) + (1 - x_1)(3x_5 + x_6 + 6x_7 + 4x_8) \geqq 7 \tag{2.26}$$

$$2x_1 + 4x_2 + 2x_3 + 4x_4 + x_5 + 5x_6 + 3x_7 + 2x_8 \geqq 9 \tag{2.27}$$

$$x_j \in \{0, 1\} \quad (j = 1, 2, \ldots, 8) \tag{2.28}$$

As we are now interested in the optimal solution of (2.25)–(2.28), as soon as a feasible solution of the problem is found then only those of better objective func-

TABLE 2.4

No.	Pseudo-solution	Feasible solution x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8	z_0 (= 9 at start)	Notes (applied rules)
1	$1234\overline{5678}$	1 1 1 1 0 0 0 0	4	c; a; b; (27)
2	$123\overline{4}$			b : (27)
3	$12\overline{3}4$			b : (26); d
4	$1\overline{2}34$			b : (27); d
5	$1\overline{23}4$			b : (26); d
6	$\overline{1}\overline{5678}\overline{234}$	0 0 0 0 1 1 1 0	3	c; a
7	$\overline{1}56\overline{7}$			b : (29)
8	$\overline{1}5\overline{6}7$			b : (29)
9	$\overline{1}5\overline{678}$			b : (29); d
10	$\overline{1}567$			b : (29)
11	$\overline{1}5\overline{67}\overline{8}$			b : (26); d
12	$\overline{1}5\overline{678}$			b : (29); d
13	$\overline{1}\overline{5678}$			b : (26); d

TABLE 2.5

No.	Pseudo-solution	Optimal solutions x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8	Notes (applied rules)
1	15678234	0 0 0 0 1 1 1 0	
2	15678		d; b : (27), (30)
3	15678		b : (27), (30)
4	156782		b : (27), (30)
5	1567823		b : (27), (30)
6	15678234		d; b : (27), (30)
7	156782		8 : d; b : (27), (30)
8	1567823		b : (27), (30)
9	15678234		(27)
10	15678234	0 0 0 0 0 1 1 1	c; a
11	15678234	0 1 0 0 0 1 1 0	
12	15678234	0 0 1 0 0 1 1 0	
13	15678234	0 0 0 1 0 1 1 0	
14	15678		d; b : (26)
15	15678234	0 1 0 0 0 0 1 1	8 : (26)
16	1567823		b : (30)
17	15678234	0 0 0 1 0 0 1 1	4 : (27)
18	15678		b : (26)

tion value will be searched. For this reason a new, so-called objective function constraint is added to the problem

$$\sum_{j=1}^{8} x_j \leqq z_0 - 1 \tag{2.29}$$

where z_0 is the objective function value belonging to the best feasible solution found so far. At the beginning, if no feasible solution is known, then $z_0 - 1$ must be an upper bound of the objective function on the domain S or D^8, as $S \subset D^8$. In our case $z_0 = 9$ will be sufficient. On solving the problem we shall utilize our knowledge of the structure of the problem; the method can easily be extended and enriched for other moderate sized problems. Table 2.4 contains the solution of the problem using the following ideas

(a) As the objective function is monotone increasing, as soon as constraints (2.26) and (2.27) are satisfied, the rest of the variables must take the value 0.

(b) If at least one of the constraints (including the objective function constraint) cannot be satisfied in the present pseudo-solution, then a backtracking step is performed.

(c) An endeavour is made to satisfy (2.26) first, secondly (2.27) – taking care of the objective function constraint (2.29) all the time. (In our simple case no more than $z_0 - 1$ variables may be fixed at value 1.)

(d) If neither (2.26) nor (2.27) is yet satisfied and there is only one free variable having a positive coefficient in this constraint at this moment, then this variable must be fixed (and tied) at value 1 (it is underlined). The first fixed variable will be x_1.

If backtracking occurs because of point (b), the corresponding constraint number is also written in the "notes" column of Table 2.4. Similarly the use of the other points is indicated. The solution of the problem can easily be followed in the tableau.

If all optimal solutions are needed, then there are two alternatives. If $z_0 - 1$ is substituted by z_0 in the objective function constraint (2.29), then we are always looking for the feasible solutions not worse instead of the better than the best one found up to now. If alternative optima are needed after determining one optimal solution, then (2.29) is substituted by the following one

$$\sum_{j=1}^{8} x_j = z^* \tag{2.30}$$

where z^* is the optimal objective function value. This is done in Table 2.5 with $z^* = 3$.

CHAPTER 3

BRANCH-AND-BOUND ALGORITHMS

3.1 Introduction

The principle of branch-and-bound was first used for mixed integer programming by Land and Doig (1960). The first application of a branch-and-bound method for a combinatorial problem appeared in the paper of Little et. al. (1963). This work demonstrated the flexibility of the method. Since then many practical and theoretical papers have appeared on the subject. Good surveys have been written by Lawler and Wood (1966) and Balinski (1970). The next two sections of this chapter contain the principle and a general framework of branch-and-bound algorithms. After this, a detailed development is followed by the solution of the knapsack and travelling salesman problems. The linear mixed integer problem is solved by the Land and Doig method. The last section contains further developments of this method.

The branch-and-bound principle itself is a quite simple but nevertheless very useful tool for solving discrete problems. The efficiency of this type of algorithm largely depends on the details, especially on the calculation of the lower and upper bounds, on the separation of sets and, finally, on the different choice rules used for determining the next solution set to be considered and the next variable to branch on.

Efficient branch-and-bound algorithms strongly utilize the structure of the problem which means that on the one hand it is not possible to design general purpose algorithms whereas on the other hand the special structure of certain problems may be well followed – thus large problems containing difficult logical conditions can also be solved.

3.2 The branch-and-bound principle

We are given a subset S_0 of the n-dimensional Euclidean space E^n and a real function $f(x)$ of n variables, which is bounded from below on the set S_0. Consider the following problem

$$\min \{f(\mathbf{x}) \mid \mathbf{x} \in S_0\} . \tag{3.1}$$

No further restrictions are imposed at present on set S_0 nor on function $f(x)$. In this book the branch-and-bound principle will be used to solve problems of discrete programming; the principle may however be used to solve problems of other types, e.g. nonlinear programming problems. The branch-and-bound principle may be considered as a tool for decomposing the domain of optimization. Furthermore,

let us suppose that we have a method for obtaining a lower bound, $L(S_i)$ of the function $f(x)$ on any subset $S_i \subset S_0$, that is,

$$K < L(S_i) \leqq f(\mathbf{x}) \quad \text{for any} \quad \mathbf{x} \in S_i \tag{3.2}$$

as the function $f(\mathbf{x})$ is bounded from below on the set S_0. Sometimes a similar upper bound, $U(S_i)$ may also be used.

$$\infty \geqq U(S_i) \geqq f(\mathbf{x}) \quad \text{for any} \quad \mathbf{x} \in S_i . \tag{3.3}$$

As the function is not supposed as being bounded from above, the upper bound function may be infinite on some subsets. For the time being only the lower bound $L(S_i)$ will be used. With regard to its real calculation, see the notes at the end of the present section and the solution of problems in the following sections.

In addition, a decomposition rule $\mathcal{C}$ is given which divides any set $S_i \subset S$ into a finite number of subsets. These subsets are preferably – but not necessarily – disjunctive sets. The decomposition rule most be closely related to the calculation of the lower bounds in order to separate the good solutions from those having high objective function values. In other words the way of lower bound calculations and the decomposition rule are to be determined together and the efficiency of the algorithms is largely dependent on them.

Now, the branch-and-bound principle may be described as follows. Let us divide the domain of optimization, S_0 using the decomposition rule $\mathcal{C}$

$$S_0 = \bigcup_{i \in I(1)} S_i \tag{3.4}$$

where $I(1)$ contains a finite number of subscripts, say

$$I(1) = \{1, 2, \ldots, r_1\}. \tag{3.5}$$

Then the lower bound of function $f(\mathbf{x})$ is determined on each of these subsets: $L(S_i)$ $(i = 1, 2, \ldots, r_1)$. Now the set S_0 itself may be discarded as it is substituted by the sets S_i, $i \in I(1)$.

After the kth subdivision, the subscript of sets to be considered will be denoted by $J(k)$. Now

$$J(1) = \{1, 2, \ldots, r_1\}$$

because of equations (3.4) and (3.5). The best one of the active (not yet discarded) sets is now determined, according to its lower bounds

$$L(S_{\nu(2)}) = \min \{L(S_i) \mid i \in J(1)\} .$$

This set, $S_{\nu(2)}$† is again subdivided into subsets by rule $\mathcal{C}$

$$S_{\nu(2)} = \bigcup_{i \in I(2)} S_i .$$

† $\nu(2)$ is used as a subscript because, $\nu(1) = 0$ is considered as the first (obligatory) choice.

The set $S_{v(2)}$ is substituted by its subsets therefore the subscripts of active sets after the second division

$$J(2) = J(1) \cup I(2) - \{v(1)\}$$

and the process is continued, by determining the set

$$S_{v(3)} : L(S_{v(3)}) = \min \{L(S_i) \mid i \in J(2)\}.$$

When the lower bound of a set S_i is determined, an attempt is made either to obtain the solution of the subproblem

$$\min \{f(\mathbf{x}) \mid \mathbf{x} \in S_i\} = f(\mathbf{x}^i) \tag{3.6}$$

or to show that subset $S_i = \emptyset$. If either of them is successful, then subset S_i will never be divided again. If $S_i = \emptyset$, then its subscript may obviously be omitted from the present set $J(k)$ and thus from any further consideration. If the optimum on subset S_i, $\mathbf{x}^i$ is determined, and the set S_i is the set to be subdivided at any time (i.e. it has the lowest lower bound among those of the active sets), then this is an indication of the fact that the optimum of the original problem (3.1) is $\mathbf{x}^i$, as will be shown later. Now a general framework of the branch-and-bound algorithms may be given for solving (3.1). Consider any decomposition rule $\mathcal{C}$ described above and any method for determining lower bounds $L(S_i)$ that satisfies inequality (3.2). Suppose that during this calculation sometimes $S_i = \emptyset$ is recognized and sometimes (3.6) is solved.

Notation

$I(k)$	subscripts of new sets emerging at branching (decomposition) k
$J(k)$	subscripts of active sets after branching k (these are the only ones to be considered)
$\hat{\mathbf{x}}$	best feasible solution found so far (optimal solution at the end)
$\hat{z} = f(\hat{\mathbf{x}})$	$(= +\infty$ if no feasible solution is found yet)
k^*	subscript of set S_i, containing the best feasible solution found up to now.
E_k	subscripts of solution sets found to be empty
P_k	subscripts of solution sets for which the problem is exactly solved
G_k	subscripts of solution sets that may be omitted because of the new feasible solution just found. (Objective function constraint.)

With the notation above, the algorithm may be summarized in the following steps. (In the description of any algorithm in this book if the jumping condition is not satisfied, automatically the following step or substep is taken.)

Step 1: Let $J(0) = \emptyset$, $v(1) = 0$, $k = 1$, $\hat{z} = +\infty$, $r_1 = 0$, $k^* = -1$.

(a) If $S_0 = \emptyset$ go to Step 8.

(b) If the optimum of problem (3.1), $\mathbf{x}^\circ$ is determined, then let $\hat{\mathbf{x}} = \mathbf{x}^\circ$, $k^* = 1$, $\hat{z} = f(\mathbf{x}^\circ)$ and go to Step 9.

Step 2: Using rule $\mathcal{C}$ determine

$$S_{v(k)} = \bigcup_{i \in I(k)} S_i, \quad I(k) = \{r_k + 1, r_k + 2, \ldots, r_{k+1}\}. \tag{3.7}$$

Step 3: Form the following sets

$E_k = \{i \mid i \in I(k), S_i = \emptyset\}$ (if discovered)

$P_k = \{i \mid i \in I(k),$ (3.6) is solved for $S_i\}$.

Step 4: If $P_k \neq \emptyset$, then let

$f(\mathbf{x}^{j(k)}) = \min \{f(\mathbf{x}^i) \mid i \in P_k\}$

and go to Step 5, otherwise $F_k = G_k = \emptyset$ and go to Step 6.

Step 5: If $f(x^{j(k)}) \geqq \hat{z}$, then $F_k = P_k$, $G_k = \emptyset$ and go to Step 6, otherwise let $\hat{x} = x^{j(k)}$, $\hat{z} = f(\hat{x})$, $F_k = P_k$, $k^* = j^{(k)}$. Form the set

$$G_k = \{i \mid i \in J(k-1) \cup I(k) - E_k - P_k, \quad L(S_i) \geqq \hat{z}\} \tag{3.8}$$

and go to the next step.

Step 6: Let $J(k) = J(k-1) \cup I(k) - E_k - F_k - G_k - \{v(k)\}$.
If $J(k) = \emptyset$ then go to Step 7, otherwise determine

$$L(S_{v(k+1)}) = \min \{L(S_i) \mid i \in J(k)\}. \tag{3.9}$$

Increase k by 1. If $L(S_{v(k)}) < \hat{z}$, then go to Step 2, otherwise go to Step 9.

Step 7: If $k^* = -1$ go to Step 8, otherwise go to Step 9.

Step 8: No solution of (3.1). Stop.

Step 9: Optimal solution of (3.1) is $\hat{\mathbf{x}} \in S_{k^*}$ with objective function value $\hat{z}$. Stop.

Until now we have not supposed that (3.1) is a discrete problem. To ensure finiteness of the algorithm we have to give some further conditions. For the sake of simplicity, at present we shall discuss only the discrete problems of finite elements. For problems of other types see the notes at the end of this section. The following two assumptions are obviously sufficient.

(A1) S_0 contains a finite number of elements.

(A2) The decomposition rule $\mathcal{C}$ divides any nonempty set $S_i \subset S_0$ into at least two nonempty subsets if set S_i contains at least two elements, and gives an indication of indivisibility if it contains a single element.

Under Assumptions (A1) and (A2) the finiteness of the algorithm would be very easy. However Assumption (A2) seems to be very strong and impractical because, in most cases, the sets S_i are implicitly given, i.e. they are determined by algebraic and logical conditions. Therefore it is usually difficult to determine whether a set S_i is empty or not. For this reason Assumption (A2) is substituted by the following one:

(A3) We are given a finite set $T_0 \supset S_0$. The decomposition rule $\mathcal{C}$ is defined on the set T_0 in such a way that it divides any nonempty set $T_i \subset T_0$ into at

least two nonempty subsets if set T_i contains at least two elements, and gives an indication of indivisibility if it contains a single element. Then the sets S_k are defined by

$$S_k = T_k \cap S_0 \tag{3.10}$$

and whenever a set S_i is to be divided then the corresponding T_i is to be decomposed and the subsets T_k define the new set S_k of the algorithm by expression (3.10).

Example: Consider the set

$$S_0 = \{\mathbf{x} \mid \mathbf{x} \in E^n, \quad \mathbf{Ax} \geqq \mathbf{b}, \quad x_j \in \{0, 1\} \quad (j = 1, 2, \ldots, n)\}$$

where $\mathbf{A}$ is an $m \times n$ matrix and $\mathbf{b}$ is a vector of m components. Then let

$$T_0 = \{\mathbf{x} \mid \mathbf{x} \in E^n, \quad x_j \in \{0, 1\} \quad (j = 1, 2, \ldots, n)\}$$

and if

$$T_k = \{\mathbf{x} \mid \mathbf{x} \in T_0, \quad x_j = \delta_j \quad \text{for all} \quad j \in M(k)\}$$

where the numbers δ_j are given constants (either 0 or 1) and the set $M(k)$ is the set of fixed variables. Then the subsets of T_k may be defined for example as

$$T_{k+1} = \{\mathbf{x} \mid \mathbf{x} \in T_k, \quad x_r = 0\}$$

$$T_{k+2} = \{\mathbf{x} \mid \mathbf{x} \in T_k, \quad x_r = 1\}$$

where $r \notin M(k)$. In other words, one of the free variables takes on its permitted different values. The variable that is to be fixed, is determined according to the constraints and the objective function of the problem. This method will often be used.

It is easily seen that rules (A1) and (A3) ensure the finiteness of the algorithm because we can return to Step 2 at most $|T_0|$ times, where $|T_0|$ is the cardinality of set T_0.

The correctness of the algorithm is stated in the following

Theorem 3.1: *If the above algorithm for problem* (3.1) *terminates in a finite number of steps (which may be guaranted, e.g. by Assumptions* (A1) *and* (A3) *for discrete problems) then*

1° *Step* 8 *may be reached only if there is no solution of* (3.1)

2° *Otherwise* $\hat{\mathbf{x}}$ *is an optimal solution of* (3.1) *with objective function value* $\hat{z}$.

Proof: First of all, it will be proved that after branching k the union of active sets

$$Q(k) = \bigcup_{r \in J(k)} S_r \tag{3.11}$$

still contains at least one optimal solution of (3.1) if $\hat{\mathbf{x}}$, the best solution found up to now, is not yet optimal. We shall show this by induction. Once an optimal solu-

tion is found, it is obviously never discarded. Therefore, in our inductive assumption we suppose that for an optimal solution $\tilde{\mathbf{x}}$

$$\tilde{\mathbf{x}} \in Q(k-1) = \bigcup_{r \in J(k-1)} S_r$$

but no optimal solution is found yet, that is,

$$f(\tilde{\mathbf{x}}) < f(\hat{\mathbf{x}})$$

where $\hat{\mathbf{x}}$ is the best solution found up to now. Now, let us consider the definition of the set $J(k)$ given in Step 6 of the algorithm.

$$\bigcup_{r \in J(k-1)} S_r = \bigcup_{r \in \bar{J}} S_r$$

where

$$\bar{J} = J(k-1) \cup I(k) - \{v(k)\}$$

and

$$\bigcup_{r \in E_k} S_r = \emptyset .$$

The set F_k is defined in Steps 4 and 5, respectively, from the set P_k. All the sets S_r $(r \in P_k)$ are discarded only if they contain no better solution, than the best one found till now, i.e. they contain no optimal solution.

In the opposite case the sets S_r $(r \in P_k)$ may or may not contain an optimal solution, but in either case one of the best solutions in these sets is saved, if it is better than $\hat{\mathbf{x}}$, the best solution found so far according to Steps 4 and 5. Similarly, the sets S_r $(r \in G_k)$ may be left out of further consideration as they cannot contain a better feasible solution than $\hat{\mathbf{x}}$. In other words, at least one optimal solution is still in the set $Q(k)$, or $\hat{\mathbf{x}}$ is already optimal.

Secondly, if the algorithm terminates with a best feasible solution $\hat{\mathbf{x}}$, it is an optimal solution of problem (3.1). Namely, suppose that branching k was the last one. There are two ways to get to Step 9. If $J(k) = \emptyset$ in Step 6 and $k^* \neq -1$, then it indicates that there is no better solution than $\hat{\mathbf{x}}$ because all the solution sets have already been eliminated. The same is true if the minimum in expression (3.9) is

$$\min \{L(S_i) \mid i \in J(k)\} \geqq \hat{z} . \tag{3.12}$$

The above reasoning also implies that Step 8 may be reached only if there is no feasible solution of the problem.

3.3 Problem solving by the branch-and-bound principle

In the previous section the branch-and-bound principle was introduced and a general framework was given for this type of algorithm. The purpose of this section is to demonstrate the use of this method in more detail.

First of all the decomposition rule and the computation of bounds must be given. They must be fitted to the type of problem they are used for. The advantage of this approach is that the special structure of the problem may be highly utilized admitting problems of large size. On the other hand there is little hope of general purpose algorithms for problems which are of really large size, say for those of several thousands of variables.

Let us try to compare two pairs of decomposition rules and bound computations $(\mathcal{C}_1, L_1)$ and $(\mathcal{C}_2, L_2)$. For the sake of simplicity, suppose that a set is always divided into two disjunctive subsets. Let us consider a set S_k and its subdivision by the two different rules

$$S_k = S^1_{k+1} \cup S^1_{k+2} = S^2_{k+2} \cup S^2_{k+2}.$$

If

$$|S^1_{k+1}| < |S^2_{k+1}|$$

and

$$L_1(S^1_{k+2}) - L_1(S^1_{k+1}) > L_2(S^2_{k+2}) - L_2(S^2_{k+1}) > 0,$$

then the rule $(\mathcal{C}_1, L_1)$ seems to be better.

On the one hand, the set of "better solutions" is smaller and thus provides a smaller set for further search, on the other hand, the distance between the two sets – according two their lower bound's – is bigger in division 1. The latter makes more likely the fact that the worse set will never have to be considered again for further division.

The above reasoning is true, however, only if both lower bound computations have the same order of accuracy, which may be measured by the difference between the real minimum of the set in question and the lower bound.

There are two opposing requirements for the calculation of lower bounds. Because it is used a great many times it must take a short time. On the other hand, it must be accurate in the above sense in order to "guide" the algorithm in a good direction. A compromise must be found when the actual computer program is developed. In most cases, a good and fast lower bound calculation can be obtained by solving a relaxed problem, i.e. a problem having a comparatively large, but nonetheless simple kind of domain. For example, some of the restrictions are omitted or several inequalities are substituted by their nonnegative linear combination, etc. This problem is of great importance and will be discussed later in more detail.

The next question is how to solve the subproblems

$$\min_{x \in S_i} f(\mathbf{x}). \tag{3.13}$$

This task seems to be similar to the original problem (3.1), but the set S_i may be very different from S_0. For example if S_i contains only a single element, then it is also the optimal solution of S_i. It is also possible that set S_i contains only a few free variables because most of the original variables already have fixed values, or the structure of set S_i may be simple. When constructing the algorithm, this point

of view is to be taken into account. If (3.13) is still too complicated, then only the lower bound $L(S_i)$ is calculated.

Similarly an effort is made to show that $S_i = \emptyset$, as many times as possible, taking into account the time necessary for these calculations. This problem has already been mentioned in Chapter 2 when we wished to determine explicitly the elements of implicitly given sets. Different tests are used for this purpose. They will be discussed in more detail in Chapter 5.

Sometimes the exact optimal solution is not necessary, only a solution of the objective function which is not further from the optimum than $\varepsilon > 0$. This is called the ε-optimal solution. The only change to be made in the algorithm is that each time a new (and better) feasible solution $\hat{\mathbf{x}}$ is found the value of the new objective function bound is

$$\hat{z} = f(\hat{\mathbf{x}}) - \varepsilon \tag{3.14}$$

instead of $f(\hat{\mathbf{x}})$. If ε is not too small this may speed up the procedure.

Besides the lower bounds, sometimes upper bounds may be used for two different purposes. First of all, if it turns out for any set S_i that

$$L(S_i) > U(S_i)$$

where

$$U(S_i) \geqq \min_{x \in S_i} f(\mathbf{x})$$

then obviously $S_i = \emptyset$. Secondly if $S_i \neq \emptyset$, then the sets S_k for which

$$L(S_k) > U(S_i)$$

may be discarded.

Alternative optima may be computed by searching all sets having the same lower bounds as the optimum of the problem.

If a computer run is terminated before the end, an estimation of the optimum and usually some good feasible solutions are available, but for this purpose some modifications will be necessary.

It should be noted that since the memory requirement of this method is usually excessive, combination with other methods, such as with single branch enumerations, is advisable. This will be discussed later.

In the following sections we shall use the branch-and-bound principle for different kinds of discrete programming problems.

The present section will probably be clearer after reading the whole chapter, because its ideas will be discussed in more detail at different problems.

3.4 Solution of the knapsack problem

Now, the knapsack problem (described in Chapter 1) will be solved by a branch-and-bound method. The problem is formulated as follows

$$(\max)\, z = \sum_{j=1} c_j x_j$$

$$\sum_{j=1}^{n} a_j x_j \leqq K \tag{3.15}$$

$$x_j \in \{0, 1\} \quad (j = 1, 2, \ldots, n)$$

where $K > 0$, $c_j > 0$, $a_j > 0$ are given integers. It is supposed that the variables are already preordered

$$\frac{c_1}{a_1} \geqq \frac{c_2}{a_2} \geqq \ldots \geqq \frac{c_n}{a_n}. \tag{3.16}$$

To apply the branch-and-bound principle a decomposition rule $\mathcal{C}$ and a method for calculating the upper bounds $U(S_i)$, should be given. (As our problem is a maximization instead of a minimization, the lower bounds should be substituted by upper bounds.)

The algorithm will be described in a very simple form. Now the set of feasible solutions is

$$S_0 = \{\mathbf{x} \mid \mathbf{x} \in D^n, \quad \sum_{j=1}^{n} a_j x_j \leqq K\}.$$

Finiteness is guaranteed by Assumption (A1) and (A3) of Section 3.2. The corresponding sets: $T_0 = D^n$, the set of all n-dimensional variables of 0–1 components; T_k is defined by fixing a free variable in one of the sets T_r, $r < s$, just as in the example following Assumption (A3). Thus the sets S_k are defined by equation (3.10). Namely,

$$S_1 = \{\mathbf{x} \mid \mathbf{x} \in S_0, \quad x_1 = 0\} \quad \text{and} \quad S_2 = \{\mathbf{x} \mid \mathbf{x} \in S_0, \quad x_1 = 1\}.$$

Let us now consider an arbitrary set of this type

$$S_k = \{\mathbf{x} \mid \mathbf{x} \in S_0, \quad x_1 = \delta_1, \ldots, x_s = \delta_s\} \tag{3.17}$$

where $\delta_1, \ldots, \delta_s$ are given constants of value 0 or 1. (The set S_k is nothing but a pseudo-solution of problem (3.15), discussed in Chapter 2.)

The upper bound $U(S_k)$ is given as the upper bound of the following problem

$$(\max)\, z = \sum_{j=1}^{s} c_j \delta_j + \sum_{j=s+1}^{n} c_j x_j$$

$$\sum_{j=s+1}^{n} a_j x_j \leqq K - \sum_{j=1}^{s} a_j \delta_j, \tag{3.18}$$

$$x_i \in \{0, 1\} \quad (j = s+1, s+2, \ldots, n)$$

defined by **Lemma 1. 2**

$$U(S_k) = \sum_{j=1}^{s} c_j \delta_j + \sum_{j=s+1}^{q} c_j + \left(K - \sum_{j=1}^{s} a_j \delta_j - \sum_{j=s+1}^{q} a_j\right) \frac{c_{q+1}}{a_{q+1}}, \tag{3.19}$$

where the subscript q is determined by the following inequalities

$$\sum_{j=s+1}^{q} a_j \leqq K - \sum_{j=1}^{s} a_j\delta_j < \sum_{j=s+1}^{q+1} a_j . \tag{3.20}$$

Note: If the first of the inequalities (3.20) is satisfied as an equation, then the upper bound is the exact optimum of (3.18) and the corresponding optimal solution is

$$x_{s+1} = \ldots = x_q = 1 \quad \text{and} \quad x_{q+1} = \ldots = x_n = 0$$

according to Lemma 1.1.

For simplicity we shall use only two tests. If, in a set S_k, defined by (3.17)

$$a_q > K - \sum_{j=1}^{s} a_j\delta_j, \quad (q > s), \tag{3.21}$$

then before calculating $U(S_k)$, S_k is substituted by

$$S_k' = \{\mathbf{x} \mid \mathbf{x} \in S_0, \quad x_1 = \delta_1, \ldots, \quad x_s = \delta_s, \quad x_q = 0 \text{ for } q \in Q\}$$

because obviously $S_k - S_k' = \emptyset$. If $Q = \{s+1, s+2, \ldots, n\}$, then S_k' contains only one feasible solution, which is the optimal solution in S_k' as well (Test T1).

If, in a set S_k, defined by (3.17)

$$\sum_{j=s+1}^{n} a_j \leqq K - \sum_{j=1}^{s} a_j\delta_j , \tag{3.22}$$

then obviously the optimal solution of (3.18) is

$$x_{s+1} = x_{s+2} = \ldots = x_n = 1$$

thus the upper bound of this set will be attained (Test T2).

The procedure described in Section 3.2 can be applied with almost no change. Instead of the minimum of lower bounds we have to determine the maximum of upper bounds in each iteration.

Let us now solve the following numerical example:

$$(\max)\ z = 30x_1 + 19x_2 + 13x_3 + 38x_4 + 20x_5 + 6x_6 + 8x_7 + 19x_8 + 10x_9 + 11x_{10}$$

$$15x_1 + 12x_2 + 9x_3 + 27x_4 + 15x_5 + 5x_6 + 8x_7 + 20x_8 + 12x_9 + 15x_{10} \leqq 62$$

$$x_j \in \{0, 1\} \quad (j = 1, 2, \ldots, 10).$$

As all the coefficients are positive integers and the variables are already in the required order (3.16) no change in the problem is necessary.

The process of solution may be followed in Figure 3.1. The circles represent solution sets of type (3.17). Inside the circle the subscript of the newly fixed variable can be found with or without a bar depending on whether its value is 0 or 1. The upper bound of the set is written beside each circle, unless further variables are

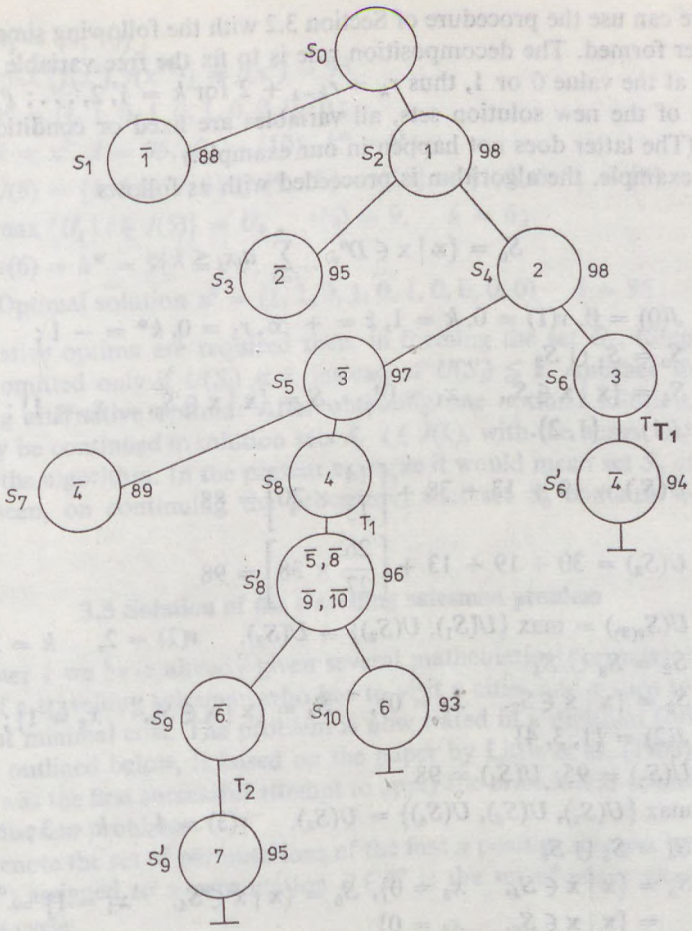


Fig. 3.1.

fixed by test T1 or T2. In this case, the upper bounds are calculated only after this fixing. The horizontal line after any solution set S_k means that no more branching is possible or necessary. This may happen in the following three cases:

1. The solution set S_k contains only a single feasible solution.
2. The solution set S_k contains no feasible solution. (3.23)
3. One of the best feasible solutions of set S_k is known.

In our simple procedure only Case 1, will occur regularly, and sometimes Case 3, if condition (3.22) is satisfied. For more complicated problems we shall try to give algorithms supplying Cases 1–3 as many times as possible without an excessive amount of computation.

Now we can use the procedure of Section 3.2 with the following simplifications: E_k is never formed. The decomposition rule is to fix the free variable of smallest subscript at the value 0 or 1, thus $r_k = r_{k-1} + 2$ for $k = 1, 2, \ldots$; $P_k = \emptyset$ only if, in one of the new solution sets, all variables are fixed or condition (3.22) is satisfied. (The latter does not happen in our example.)

In our example, the algorithm is proceeded with as follows

$$S_0 = \{\mathbf{x} \mid \mathbf{x} \in D^n, \quad \sum_{j=1}^{n} a_j x_j \leqq k\}.$$

Step 1: $J(0) = \emptyset$, $v(1) = 0$, $k = 1$, $\hat{z} = +\infty$, $r_1 = 0$, $k^* = -1$;

Step 2: $S_0 = S_1 \cup S_2$

$S_1 = \{\mathbf{x} \mid \mathbf{x} \in S_0, \quad x_1 = 0\}$, $\quad S = \{\mathbf{x} \mid \mathbf{x} \in S_0, \quad x_1 = 1\}$;

Step 6: $J(1) = \{1, 2\}$.

$$U(S_1) = 19 + 13 + 38 + \left[\frac{14}{15} \times 20\right] = 88$$

$$U(S_2) = 30 + 19 + 13 + \left[\frac{26}{27} \times 38\right] = 98$$

$U(S_{v(2)}) = \max\{U(S_1), U(S_2)\} = U(S_2)$, $\quad v(2) = 2$, $\quad k = 2$;

Step 2: $S_2 = S_3 \cup S_4$

$S_3 = \{\mathbf{x} \mid \mathbf{x} \in S_2, \quad x_2 = 0\}$, $\quad S_4 = \{\mathbf{x} \mid \mathbf{x} \in S_2, \quad x_2 = 1\}$;

Step 6: $J(2) = \{1, 3, 4\}$

$U(S_3) = 95$, $U(S_4) = 98$

$\max\{U(S_1), U(S_3), U(S_4)\} = U(S_4)$, $\quad v(3) = 4$, $\quad k = 3$;

Step 2: $S_4 = S_5 \cup S_6$

$S_5 = \{\mathbf{x} \mid \mathbf{x} \in S_4, \quad x_3 = 0\}$, $S_6 = \{\mathbf{x} \mid \mathbf{x} \in S_4, \quad x_3 = 1\} =$

$= \{\mathbf{x} \mid \mathbf{x} \in S_6, \quad x_4 = 0\}$

(Applying Test T1);

Step 6: $J(3) = \{1, 3, 5, 6\}$

$U_5 = U(S_5) = 97 \quad U_6 = U(S_6) = 94$

$\max\{U_i \mid i \in J(3)\} = U_5 \quad v(4) = 5, \quad k = 4$;

Step 2: $S_5 = S_7 \cup S_8$

$S_7 = \{\mathbf{x} \mid \mathbf{x} \in S_5, \quad x_4 = 0\}$ $S_8 = \{\mathbf{x} \mid \mathbf{x} \in S_5, \quad x_4 = 1\} =$

$= \{\mathbf{x} \mid \mathbf{x} \in S_8, \quad x_j = 0 \quad \text{for } j = 5, 8, 9, 10\}$ (Test T1);

Step 6: $J(4) = \{1, 3, 6, 7, 8\}$ $U_7 = 89$ $U_8 = 96$

$\max\{U_i \mid i \in J(4)\} = U_8 \quad v(5) = 8, \quad k = 5$;

Step 2: $S_8 = S_9 \cup S_{10}$

$S_9 = \{\mathbf{x} \mid \mathbf{x} \in S_8, \quad x_6 = 0, \quad x_7 = 1\}$ (Test T2)

$S_{10} = \{\mathbf{x} \mid \mathbf{x} \in S_8, \quad x_6 = 1, \quad x_7 = 0\}$ (Test T1);

Step 3: $P_5 = \{9, 10\}$;

Step 4: $\max \{f(\mathbf{x}^9), f(\mathbf{x}^{10})\} = f(\mathbf{x}^9) = 95$

$x^9 = (1, 1, 0, 1, 0, 1, 0, 0, 0, 0)$;

Step 5: $\hat{x} = x^9$, $\hat{z} = 95$, $F_k = \{10\}$, $k^* = 9$;

Step 6: $J(5) = \{1, 3, 6, 7, 8\} \cup \{9, 10\} - \{10\} - \{1, 3, 6, 7\} - \{8\} = \{9\}$.

$\max \{U_i \mid i \in J(5)\} = U_9 \quad v(6) = 9, \quad k = 6$;

Step 7: $v(6) = k^* = 9$;

Step 9: Optimal solution $x^9 = (1, 1, 0, 1, 0, 1, 0, 0, 0, 0) \quad \hat{z} = 95$.

If alternative optima are required then, in forming the set G_k, (Step 5) set S_i should be omitted only if $U(S_i) < \hat{z}$, instead of $U(S_i) \leqq \hat{z}$. Another method for determining alternative optima: After obtaining one optimal solution, the procedure may be continued in solution sets S_i $i \in J(k)$, with the actual value of k at the end of the algorithm. In the present example it would mean set S_3 only. It can easily be seen, on continuing the procedure, that set S_3 contains no optimal solution.

3.5 Solution of the travelling salesman problem

In Chapter 1 we have already given several mathematical formulations for the problem of a travelling salesman who has to visit n cities and return to the starting point at minimal cost. The problem is now stated in a different form and the algorithm, outlined below, is based on the paper by Little et al. (1963). Actually this paper was the first successful attempt to apply the branch-and-bound principle to a pure discrete problem.

Let us denote the set of permutations of the first n positive integers by $\mathcal{S}^n$. Then, a tour $T(P)$ assigned to a permutation $P \in \mathcal{S}^n$ is the set of edges along the corresponding cycle

$$T(P) = \{(i_1, i_2), (i_2, i_3), \ldots, (i_{n-1}, i_n), (i_n, i_1)\}$$

where

$$P = (i_1, i_2, \ldots, i_n).$$

As in Models 2 and 3 of Chapter 1, the travelling cost c_{ij} from City i to City j, is supposed as being nonnegative integer for any $i \neq j$, and $c_{ii} = \infty$. (This assumption does not restrict the generality, it is made only for simplicity.) Similarly as before

$$\mathbf{x}_{ij} = \begin{cases} 1 \text{ if City } j \text{ is visited immediately after City } i \\ 0 \text{ otherwise.} \end{cases}$$

The problem is the following

$$(\min)\ z = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{3.24}$$

on the set

$$S_0 = \{\mathbf{x} \mid \mathbf{x} \in D^{n^2},\ \exists P \in \mathfrak{S}^n : x_{ij} = 1 \Leftrightarrow (i,j) \in T(P)\}\dagger \tag{3.25}$$

that is, only the vectors $\mathbf{x} \in D^n$ are considered, in order to correspond to a single tour.

It is obvious that the conditions

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad (j = 1, \ldots, n) \tag{3.26}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad (i = 1, \ldots, n) \tag{3.27}$$

$$x_{ij} \in \{0, 1\} \quad (i, j = 1, 2, \ldots, n) \tag{3.28}$$

are satisfied for any vector $\mathbf{x} \in S_0$.

Although symmetricity $c_{ij} = c_{ji}$ or the triangle inequality

$$c_{ij} + c_{jk} \geqq c_{ik}$$

may speed up the algorithm using special tests, for the time being, we shall not use these assumptions and we can thus keep the discussion as simple and as general as possible.

If we wish to use the branch-and-bound principle for our problem, we have to give a decomposition rule for subsets of the set S_0 of feasible solutions, and a method for obtaining good lower bounds of the objective function (3.24) of the subsets so determined.

The usual kind of decomposition will be applied for a set $S_p \subset S_0$

$$S_p = S'_p \cup S''_p$$
$$S'_p = \{\mathbf{x} \mid \mathbf{x} \in S_p,\ x_{rs} = 0\}$$
$$S''_p = \{\mathbf{x} \mid \mathbf{x} \in S_p,\ x_{rs} = 1\}$$

where x_{rs} is a variable not yet fixed in set S_p.

It is obvious that set S''_p contains subtantially fewer solutions in general, because the fixing $x_{rs} = 1$ implies

$$x_{rj} = x_{is} = 0 \quad \text{for any} \quad j \neq s \quad \text{and} \quad i \neq r,$$

but the fixing $x_{rs} = 0$ usually has fewer consequences. The loop conditions (which exclude the existence of several loops) may sometimes change the picture, but basically it remains the same. In other words, we have to strive to obtain better solutions (smaller lower bounds) for set S''_p than for S'_p, having as large a difference between the lower bounds as possible. Therefore, before giving a method for determining the variable x_{rs} on which we split, we have turn to the calculation of lower bounds.

After fixing any number of variables the structure of the problem does not change, only some of the costs c_{ij} are substituted by ∞ or some of the rows and

† The signs $\exists$ and $\Leftrightarrow$ stand for "there exists" and "if and only if", respectively.

columns are deleted from matrix $\mathbf{C}$ (as a consequence of a fixing $x_{rs} = 1$ for example). Therefore it is sufficient to give the calculation for the set S_0.

Constraints (3.26)–(3.28) are consequences of (3.25), thus the solution of the assignment problem with the objective function (3.24) and the constraints (3.26)–(3.28) would give a lower bound of the original problem. If, by chance, the solution of the assignment problem resulted in a single-loop solution (i.e. it would satisfy condition (3.25)) then we would obtain the optimal solution of (3.24)–(3.25). However, this is not very likely in general. On the other hand, since lower bounds must be calculated in each iteration of the algorithm, it must be simple.

The matrix reduction, used in the algorithm to solve an assignment problem, may be applied.

Lemma 3.1: *An optimal solution* $\hat{\mathbf{x}}$ *of* (3.24–(3.25) *remains optimal if the matrix of the objective function coefficients* $\mathbf{C}$ *is substituted by matrix* $\mathbf{C}'$

$$c'_{ij} = c_{ij} - g_i - h_j \quad (i, j = 1, 2, \ldots, n) \tag{3.29}$$

where g_i *and* h_j *are arbitrary integers.*

Proof: Consider any vector $\mathbf{x} \in S_0$. As $\hat{\mathbf{x}}$ is an optimal solution

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}\hat{x}_{ij} \leqq \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}x_{ij}.$$

Using equations (3.26) and (3.27)

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c'_{ij}\hat{x}_{ij} = \sum_{i=1}^{n} \sum_{j=1}^{n} (c_{ij} - g_i - h_j)\hat{x}_{ij} = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}\hat{x}_{ij} - \sum_{i=1}^{n} g_i - \sum_{j=1}^{n} h_j \leqq$$

$$\leqq \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}x_{ij} - \sum_{i=1}^{n} g_i - \sum_{j=1}^{n} h_j = \sum_{i=1}^{n} \sum_{j=1}^{n} c'_{ij}x_{ij}$$

which shows that $\hat{\mathbf{x}}$ remains optimal.

The above operation (3.29) may be called matrix reduction. It should be done in such a way that $c'_{ij} \geqq 0$, and each row and each column of matrix $\mathbf{C}'$ contains at least one zero. In this case no further reduction is possible by positive g_i or h_j. Obviously,

$$\sum_{i=1}^{n} h_i + \sum_{j=1}^{n} g_j \tag{3.30}$$

is a lower bound for the original problem, as the new objective function is always nonnegative, and it is smaller than the original objective function by precisely the sum (3.30). Thus, throughout the procedure we may get a lower bound for any set $S_p \subset S_0$, if S_p is obtained from S_0 by fixing certain variables. Only a matrix reduction should be performed on the corresponding matrix.

Now we may turn to determining the free variable x_{rs} to branch on. Again, it is sufficient to consider the original set S_0 of feasible solutions as the subproblems – obtained later – have the same form. Let us suppose that matrix $\mathbf{C}$

is already in reduced form, i.e., $c_{ij} \geqq 0$ and each row and each column contains at least one zero. The set S_0 is decomposed into disjoint subsets:

$$S_0 = S_1 \cup S_2 .$$

$$S_1 = \{\mathbf{x} \mid \mathbf{x} \in S_0, \quad x_{rs} = 0\} \quad \text{and} \quad S_2 = \{\mathbf{x} \mid \mathbf{x} \in S_0, \quad x_{rs} = 1\}.$$

In set S_2, the problem remains unchanged except that row r and column s are deleted at a cost c_{rs}. The remaining rows and columns still contain only non-negative elements, thus the lower bound for set S_2 is

$$L(S_2) = c_{rs} \tag{3.31}$$

(Sometimes further matrix reduction may be possible after deleting row r and column s. In this case $L(S_2)$ is increased by the sum of reductions.)

In set S_1, it is clear from (3.26) and (3.27) that there must be an $x_{rj} = 1$ $(j \neq s)$ and an $x_{is} = 1$ $(i \neq r)$, because $x_{rs} = 0$.

Thus

$$L(S_1) = \min_{j|j \neq s} c_{rj} + \min_{i|i \neq r} c_{is} \tag{3.32}$$

is a lower bound for increasing the objective function. We shall choose the variable x_{rs}, for which the difference of these lower bounds is maximal:

$$\max_{r,s} \{L(S_1) - L(S_2)\} . \tag{3.33}$$

It is obviously sufficient to consider the pairs (r, s), for which $c_{rs} = 0$. In the case of $c_{rs} > 0$, namely,

$$L(S_2) > 0 \quad \text{and} \quad L(S_1) = 0$$

the latter holds true because row r and column s contain a zero. On the other hand, if $c_{rs} = 0$, then

$$L(S_2) = 0 \quad \text{and} \quad L(S_1) \geqq 0.$$

The above reasoning holds true also when any later solution set S_i is decomposed into sets S_r and S_{r+1}, but the increase of lower bounds $L(S_r) - L(S_i)$ and $L(S_{r+1}) - L(S_i)$ should be considered instead of the lower bounds $L(S_1)$ and $L(S_2)$ respectively. When their difference is formed the superfluous term $L(S_i)$ disappears.

This means that instead of (3.33) the following simple maximization should be made to determine the branching variable x_{rs}

$$\max_{(r,s)|c_{rs}=0} (\min_{j \neq s} c_{rj} + \min_{i \neq s} c_{is}) . \tag{3.34}$$

If the examined set is not S_0, but

$$S_p = \{\mathbf{x} \mid \mathbf{x} \in S_0, \quad x_{i_1 j_1} = \delta_1, \; x_{i_2 j_2} = \delta_2, \ldots x_{i_k j_k} = \delta_k\} ,$$

then the choice rule is basically the same, but some changes may be necessary. First of all, it is possible that there are obligatory choices. For example, if

$$x_{uj} = 0 \quad j = 1, \ldots, v-1, \quad v+1, \ldots, n ,$$

then obviously

$$x_{uv} = 1$$

is necessary. The same is true for columns. Another kind of obligatory fixing arises after each fixing level 1, say $x_{rs} = 1$. Then x_{sr} must be 0. Furthermore, if the longest path in S_p containing the edge (r, s) consists of at least two and at most $n - 2$ edges, that is

$$x_{k_1k_2} = \ldots = x_{k_tr} = x_{rs} = x_{sk_{t+1}} = \ldots = x_{k_{u-1}k_u} = 1, \quad 1 \leqq u \leqq n - 3 \tag{3.35}$$

(no edge of form $x_{ik_1} = 1$ or $x_{k_uj} = 1$), then to avoid a subtour, the fixing

$$x_{k_uk_1} = 0$$

is necessary. These fixings may be performed after the branching or they may be taken into account when searching for branching variable x_{rs}. Before applying the algorithm of Section 3.2, matrix **C** should be reduced. In each iteration in Step 6, before determining the lower bounds of the new sets, the above necessary fixings have to be performed in order to obtain more accurate lower bounds and to avoid useless branchings.

A good survey of the different models and solutions for the travelling salesman may be found in Bellmore and Nemhauser (1968). A further development resulted from Held and Karp (1971). They use minimum spanning trees to determine very good lower bounds.

Example: Solve the travelling salesman problem having the following (non-symmetric) cost matrix

	31	15	23	10	17
16		24	7	12	12
34	3		25	54	25
15	20	33		50	40
16	10	32	3		23
18	20	13	28	21	

Throughout the solution empty spaces will mean infinite costs – prohibited choices. To obtain the starting tableau C_0, the first step is to reduce each row by its minimal element and column 6 of the remaining matrix by 5, that is,

$$g_1 = 10,\ g_2 = 7,\ g_3 = 3,\ g_4 = 15,\ g_5 = 3,\ g_6 = 13,\ f_6 = 5,$$

$$L(S_0) = \sum_{i=1}^{n} g_i + \sum_{j=1}^{n} f_j = 56$$

	21	5	13	0	2				7	
9		17	0	5	0			0		2
31	0		22	51	17		22			
0	5	18		35	20	10				
13	7	29	0		15			7		
5	7	0	15	8			10			

C^0

$\min_{j \mid j \neq s} c_{rj} + \min_{i \mid i \neq r} c^0_{is}$ is for $c_{rs} = 0$

The arising branches may be seen in Figure 3.2. Now we follow the steps of the general branch-and-bound algorithm (described in section 3.2).

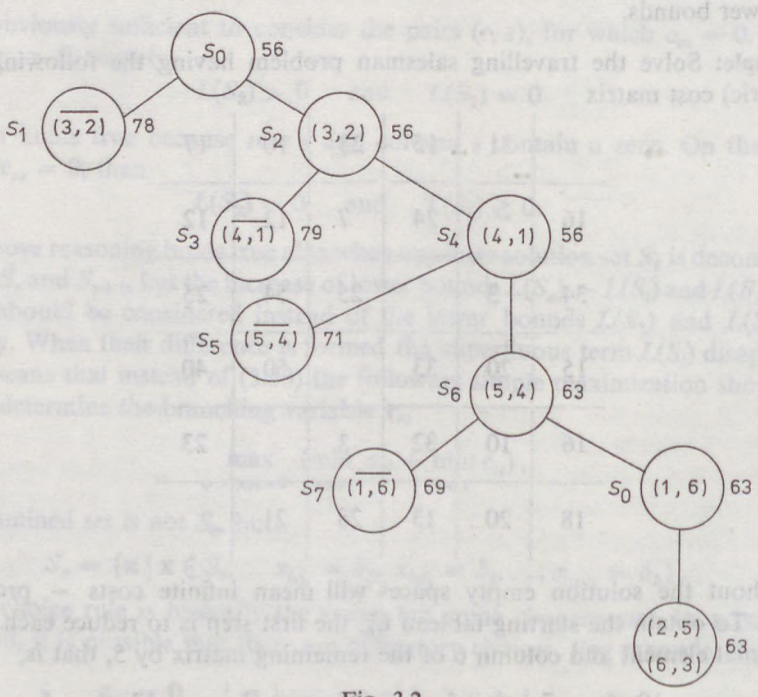


Fig. 3.2

Step 1: $J(0) = \emptyset$, $v(1) = 0$, $k = 1$, $\hat{z} = \infty$, $r_0 = 0$, $k^* = -1$.

Step 2: Using criterion (3.34), $(r, s) = (3, 2)$, thus

$$S_0 = S_1 \cup S_2$$

$S_1 = \{\mathbf{x} \mid \mathbf{x} \in S_0, \quad x_{3,2} = 0\}$, $S_2 = \{\mathbf{x} \mid \mathbf{x} \in S_0, \quad x_{3,2} = 1\}$, $I_1 = \{1, 2\}_*^1$.

Matrix $\mathbf{C}^1$ is obtained from $\mathbf{C}^0$ by substituting $c_{32} = \infty$, and reducing it.

	16	5	13	0	2
		17	0	5	0
14			5	34	0
0	0	18		35	20
13	2	29	0		15
5	2	0	15	8	

$\mathbf{C}^1$

The sum of reductions gives the lower bound increase

$$L(S_1) = L(S_0) + 17 + 5 = 78.$$

Similarly matrix $\mathbf{C}^2$ is obtained from $\mathbf{C}^0$ by omitting row 3 and column 2. To avoid a subtour: $x_{2,3} = 0$, therefore $c_{2,3} = \infty$. No reduction is possible, thus $L(S_2) = 56$.

		5	13	0	2
9			0	5	0
0		18		35	20
13		29	0		15
5		0	15	8	

$\mathbf{C}^2$

				7	
			0		2
23					
			7		
		10			

$\min\limits_{j \mid j \neq s} c_{rj}^2 + \min\limits_{i \mid i \neq r} c_{is}^2$
for 0 elements of matrix $\mathbf{C}^2$

Step 6: $J(1) = \{1, 2\}$

$\min \{L(S_i) \mid i \in J(1)\} = L(S_2)$, $v(2) = 2$, $k = 2$.

Step 2: As before, the branching variable is determined from (3.34):

$(r, s) = (4, 1)$

$S_2 = S_3 \cup S_4$ $\quad x_{4,1} = 0$ and 1 respectively.

Matrix $\mathbf{C}^3$ is obtained from $\mathbf{C}^2$ by substituting $c_{41} = \infty$, and reducing it:

		5	13	0	2
4			0	5	0
	X				
		0		17	2
8		29	0		15
0		0	15	8	

$\mathbf{C}^3$

$L(S_3) = L(S_2) + 18 + 5 = 79.$

In matrix $\mathbf{C}^4$, $x_{1,4}$ must be 0 $\quad (c^4_{1,4} = \infty)$ and row 4 and column 1 are deleted:

		5		0	2
			0	5	0
X					
		29	0		15
		0	15	8	

$\mathbf{C}^4$

$L(S_4) = 56$

				10	
			2		2
			15		
		13			

$\min_{j \mid j \neq s} c^4_{rj} + \min_{i \mid i \neq r} c^4_{is}$

Step 6: $J(2) = \{1, 3, 4\}$

$\min \{L(S_i) \mid i \in J(2)\} = L(S_4)$ $\quad \nu(3) = 4, \quad k = 3.$

Step 2: $S_4 = S_5 \cup S_6$ $\quad x_{54} = 0$ and 1 respectively.

		5	13	0	2
			0	5	0
		14			0
		0	15	8	

C^5

$L(S_5) = L(S_4) + 15 = 71.$

To exclude subtours for S_6, x_{45} and x_{15} must be 0, that is, $c^6_{45} = c^6_{15} = \infty$. Deleting row 5 and column 4, a reduction of 7 units is possible.

	3			0					3
			0	0				3	0
	0		3				6		

C^6

$L(S_6) = L(S_4) + 7 = 63$

Step 6: $J(3) = \{1, 3, 5, 6\}$

$\min \{L(S_i) \mid i \in J(3)\} = L(S_6)$ $\quad \nu(4) = 6, \quad k = 4.$

Step 2: $S_6 = S_7 \cup S_8 \quad x_{63} = 0$ and 1 respectively.

$L(S_7) = L(S_6) + 6 = 69$

Step 3: The only solution in S_8 is obviously

$$x_{25} = x_{63} = 1,$$

thus

$$\hat{\mathbf{x}} : \{x_{32} = x_{41} = x_{54} = x_{16} = x_{25} = x_{63} = 1\}$$

$$\hat{z} = 63, \; k^* = 8.$$

Step 5: $F_k = \emptyset, \; G_k = \{1, 3, 5, 7\}.$

Step 6: $J(k) = 8, \; v(5) = 8, \; k = 5.$

Step 7: $v(k) = k^*$. $\hat{\mathbf{x}}$ is an optimal solution. Stop.

The optimal tour is

$$1, 6, 3, 2, 5, 4, 1.$$

3.6 The method of Land and Doig for the solution of mixed integer problems

The method presented here is based on the work of Land and Doig (1960) which was actually the first paper applying the branch-and-bound principle.

Let us consider the following linear mixed integer programming problem:

$$(\max)\, z = \mathbf{c}^T\mathbf{x} + \mathbf{d}^T\mathbf{y} \tag{3.36}$$

$$\mathbf{G}\mathbf{x} + \mathbf{H}\mathbf{y} = \mathbf{b} \tag{3.37}$$

$$\mathbf{r} \geqq \mathbf{x} \geqq \mathbf{0} \tag{3.38}$$

$$\mathbf{y} \geqq \mathbf{0} \tag{3.39}$$

$$x_j \in \{0, 1, \ldots, r_j\} \quad (j = 1, 2, \ldots, n_1). \tag{3.40}$$

The matrices $\mathbf{G}$ and $\mathbf{H}$ of sizes $(m \times n_1)$ and $(m \times n_2)$, respectively, are given, and the vectors $\mathbf{r} = (\mathbf{r}_1, \ldots, \mathbf{r}_{n_1}) \geqq 0$ (integer), $\mathbf{b}$, $\mathbf{c}$, $\mathbf{d}$ of corresponding size are given. Let us suppose that the set defined by (3.36)–(3.39) is bounded. The set of vector pairs $(\mathbf{x}, \mathbf{y})$ satisfying the constraints (3.37)–(3.40), i.e. the set of feasible solutions will be denoted by R. The set R' is obtained from the set R by omitting the integrality condition (3.40). The problems of maximizing the objective function (3.36) on the set R and R' are denoted by F and F' respectively.

The following two lemmata will be needed in the algorithm.

Lemma 3.2: *Let us consider a pair of vectors* $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ *and a subscript* j *for which*

$$(\hat{\mathbf{x}}, \hat{\mathbf{y}}) \in R' \qquad \text{and} \qquad \hat{x}_j \neq [\hat{x}_j].$$

Then the equations

$$\{(\mathbf{x}, \mathbf{y}) \in R' \mid x_j = [\hat{x}_j]\} = \emptyset \tag{3.41}$$

$$\{(\mathbf{x}, \mathbf{y}) \in R' \mid x_j = [\hat{x}_j] + 1\} = \emptyset \tag{3.42}$$

imply, that

$$R = \emptyset . \tag{3.43}$$

Proof: For simplicity let $j = 1$, that is, $x_1 \neq [\hat{x}_1]$. Let us suppose indirectly, that the assumptions (3.41) and (3.42) hold for $j = 1$, yet there exists a point $(\hat{\mathbf{x}}, \hat{\mathbf{y}}) \in R$.

As the set R' is convex and $R' \supset R$,

$$(\lambda\hat{\mathbf{x}} + (1 - \lambda)\tilde{\mathbf{x}}, \quad \lambda\hat{\mathbf{y}} + (1 - \lambda)\tilde{\mathbf{y}}) \in R' \qquad \text{for any} \qquad 0 \leqq \lambda \leqq 1 .$$

It follows from the assumptions, that $\tilde{x}_1$ is integer and thus either $\tilde{x}_1 < [\hat{x}_1]$ or $\tilde{x}_1 > [\hat{x}_1] + 1$. But then, there exists a $0 < \lambda_1 < 1$ so that either

$$\lambda_1\hat{x}_1 + (1 - \lambda_1)\tilde{x}_1 = [\hat{x}_1]$$

or

$$\lambda_1\hat{x}_1 + (1 - \lambda_1)\tilde{x}_1 = [\hat{x}_1] + 1$$

which contradicts assumptions (3.41) and (3.42), thus the lemma is proved.

Note: It follows from the proof that if only one of the conditions (3.41) and (3.42) is satisfied, then

$$R_1 = \{(\mathbf{x}, \mathbf{y}) \mid (\mathbf{x}, \mathbf{y}) \in R, x_j \leqq [\hat{x}_j]\} = \emptyset$$

and

$$R_2 = \{(\mathbf{x}, \mathbf{y}) \mid (\mathbf{x}, \mathbf{y}) \in R, x_j \geqq [\hat{x}_j] + 1\} = \emptyset$$

respectively.

Lemma 3.3: *Let* $(\hat{\mathbf{x}}, \hat{\mathbf{y}}) \notin R$ *be an optimal solution of problem* F'. *Then let us consider a subscript* j *for which* $\hat{x}_j \neq [\hat{x}_j]$. *Furthermore, let* γ *and* δ *be arbitrary positive integers in such a way that*

$$\gamma < \hat{x}_j < \delta .$$

Notation

$$\alpha' = \max \{z \mid (\mathbf{x}, \mathbf{y}) \in R', \quad x_j = \gamma\}$$

$$\alpha'' = \max \{z \mid (\mathbf{x}, \mathbf{y}) \in R', \quad x_j = \delta\}$$

$$\beta' = \max \{z \mid (\mathbf{x}, \mathbf{y}) \in R', \quad x_j \leqq \gamma - 1\}$$

$$\beta'' = \max \{z \mid (\mathbf{x}, \mathbf{y}) \in R', \quad x_j \geqq \delta + 1\} .$$

Then, if the above assumptions hold,

$$\max \{\alpha', \ \alpha''\} \geqq \max \{\beta', \beta''\} . \tag{3.44}$$

Proof: For simplicity, let $j = 1$, that is $\hat{x}_1 \neq [\hat{x}_1]$. Instead of inequality (3.44) it is sufficient to see that

$$\alpha' \geqq \beta' \tag{3.45}$$

and $$\alpha'' \geqq \beta''. \tag{3.46}$$

In order to demonstrate inequality (3.45) let

$$\alpha' = z(\mathbf{x}', \mathbf{y}'), \quad (\mathbf{x}', \mathbf{y}') \in R' \quad \text{and} \quad x_1' = \gamma.$$

Furthermore, let us suppose indirectly, that there exists a pair of vectors $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) \in R'$ with $\tilde{x}_1 \leqq \gamma - 1$, for which

$$z(\mathbf{x}', \mathbf{y}') = \alpha' < \beta' = z(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}). \tag{3.47}$$

As $R \subset R'$ and the set R' is convex,

$$(\lambda\hat{\mathbf{x}} + (1 - \lambda)\tilde{\mathbf{x}}, \quad \lambda\hat{\mathbf{y}} + (1 - \lambda)\tilde{\mathbf{y}}) \in R' \quad \text{for any} \quad 0 \leqq \lambda \leqq 1.$$

Then the inequalities $\tilde{x}_1 < x_1' < \hat{x}_1$ imply that there exists a λ' such that

$$x_1' = \lambda'\tilde{x}_1 + (1 - \lambda')\hat{x}_1 \quad (0 < \lambda' < 1).$$

Now consider the following series of inequalities

$$z(\mathbf{x}', \mathbf{y}') \geqq \lambda' z(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) + (1 - \lambda')z(\hat{\mathbf{x}}, \hat{\mathbf{y}}) \geqq z(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$$

which contradicts the indirect assumption (3.47). Inequality (3.46) may be shown in a similar way, thus the lemma is proved.

The method will be formulated in terms of the algorithm described in Section 3.2. As the problem is maximization, upper bounds are used instead of lower bounds. Now we shall point out some differences, furthermore, the way in which the sets E_k, P_k, etc. of the algorithm will be determined in the mixed integer case.

The continuous problem F', that is, (3.36)–(3.39), is solved first. Let us denote the optimum by $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$. If all components of vector $\hat{\mathbf{x}}$ are integer, then the discrete problem F is also solved. In the opposite case let us choose one of the noninteger components, say $\hat{x}_j \neq [\hat{x}_j]$. Then the following subsets of set R' are formed:

$$R_k' = \{(\mathbf{x}, \mathbf{y}) \mid (\mathbf{x}, \mathbf{y}) \in R', \quad x_j = k\} \qquad k = 1, 2, \ldots, r_j.$$

The maximization of function z is carried out, however, only on two sets having the subscripts $k_1 = [\hat{x}_j]$ and $k_2 = [\hat{x}_j] + 1$ respectively. The optimum of the other sets cannot be higher than the optimum of the sets R_{k_1}' and R_{k_2}', according to Lemma 3.3. Naturally, if the set $R_{k_1}'(R_{k_2}')$ is shown to be empty, then the sets R_k for $k < k_1$ (for $k > k_2$) are also empty according to Lemma 3.2. On the other hand, if the set $R_{k_1}'(R_{k_2}')$ is decomposed into subsets, then the sets R_k for $k < k_1$ for $(k < k_2)$ should be implicitly bounded by determining

$$\max \{z \mid (\mathbf{x}, \mathbf{y}) \in R_{k_1 - 1}\} \qquad \text{and} \qquad \max \{z \mid (\mathbf{x}, \mathbf{y}) \in R_{k_2 + 1}\}$$

respectively. The algorithm is carried out similarly if a solution set is examined in which some variables are already fixed.

Apart from these modifications, the basic idea of the algorithm remains the same as in Section 3.2. It is not a trivial matter, however, how these changes should be included, therefore the modified algorithm will be stated.

Besides that already used in Section 3.2 the following additional **notation** will be used:

$\mu(k)$ the subscript of the variable used for branching at iteration k (at the kth decomposition)

$\omega(q)$ the subscript of the immediate predecessor of set S_q

$\tau(q)$ set S_q is obtained from $S_{\omega(q)}$ by fixing variable $x_{\tau(q)}$

$(\mathbf{x}^P, \mathbf{y}^P)$ is a continuous optimal solution of the problem of set S_p. (Integrality constraints are disregarded.):

$$\mathbf{c}^T\mathbf{x}^P + \mathbf{d}^T\mathbf{y}^P = \max_{(\mathbf{x},\mathbf{y})\in S'_p} (\mathbf{c}^T\mathbf{x} + \mathbf{d}^T\mathbf{y})$$

$$\eta(q) = \begin{cases} -1 & \text{if } x_{\tau(q)} \leqq [x^{\omega(q)}_{\tau(q)}] \\ 1 & \text{otherwise.} \end{cases}$$

Step 1: Let $J(0) = \emptyset$, $v(1) = 0$, $k = 1$, $\hat{z} = -\infty$, $k^* = -1$. Solve the problem

$$\max \{z(\mathbf{x}, \mathbf{y}) \mid (\mathbf{x}, \mathbf{y}) \in S_0\}.$$

(a) If it has no solution, go to Step 8.

(b) In the opposite case, let us denote an optimal solution of this problem by $(\mathbf{x}^0, \mathbf{y}^0)$. If $\mathbf{x}^0$ has only integer components, then $k^* = 0$, $(\hat{\mathbf{x}}, \hat{\mathbf{y}}) = (\mathbf{x}^0, \mathbf{y}^0)$, $\hat{z} = z(\hat{\mathbf{x}}, \hat{\mathbf{y}})$, $k = 0$ and go to Step 9. Otherwise let $q = 0$ and:

Step 2: Choose a noninteger component of $\mathbf{x}^q$, say $x^q_{\mu(k)}$, and solve the problems

$$\max \{z \mid (\mathbf{x}, \mathbf{y}) \in S_{q+1}\} \quad \text{and} \quad \max \{z \mid (\mathbf{x}, \mathbf{y}) \in S_{q+2}\},$$

where

$$S_{q+1} = \{(\mathbf{x}, \mathbf{y}) \mid (\mathbf{x}, \mathbf{y}) \in S_{v(k)}, \quad x_{\tau(q+1)} = \delta_{q+1}\}$$

and

$$S_{q+2} = \{(\mathbf{x}, \mathbf{y}) \mid (\mathbf{x}, \mathbf{y}) \in S_{v(k)}, \quad x_{\tau(q+2)} = \delta_{q+2}\},$$

with

$$\tau(q-1) = \tau(q+2) = \mu(k), \quad \delta_{q+1} = \delta_{q+2} - 1 = [x^q_{\mu(k)}].$$

Let

$$\omega(q+1) = \omega(q+2) = v(k)$$

$$\eta(q+1) = -1, \quad \eta(q+2) = 1.$$

The continuous optimal solutions of these problems are denoted by $(\mathbf{x}^{q+1}, \mathbf{y}^{q+1})$ and $(\mathbf{x}^{q+2}, \mathbf{y}^{q+2})$, respectively. Thus the corresponding upper bounds are:

$$U(S_{q+1}) = z(\mathbf{x}^{q+1}, \mathbf{y}^{q+1}) \qquad U(S_{q+2}) = z(\mathbf{x}^{q+2}, \mathbf{y}^{q+2}).$$

Step 3: If $q = 0$, then let $U(S_{q+3}) = -\infty$ and go to Step 4.

(a) If $\eta(v(k)) = -1$, then let $\delta_{q+3} = \delta_{v(k)} - 1$, and $\eta(q+3) = -1$. For $\delta_q = 0$ let $U(S_{q+3}) = -\infty$ and go to Step 4, otherwise go to (c)

(b) If $\eta(v(k)) = 1$, then let $\delta_{q+3} = \delta_{q+1}$, $\eta(q+3) = 1$. For $\delta_q = r_{\tau(q)}$ let $U(S_{q+3}) = -\infty$ and go to Step 4 otherwise go to (c)

(c) Solve the problem

$$\max \{z \mid (\mathbf{x}, \mathbf{y}) \in S_{q+3}\}$$

$$S_{q+3} = \{(\mathbf{x}, \mathbf{y}) \mid (\mathbf{x}, \mathbf{y}) \in S_{\omega(v(k))},\ x_{\tau(q+3)} = \delta_{q+3}\} \quad \tau(q+3) = \tau(q).$$

The optimal solution is denoted by $(\mathbf{x}^{q+3}, \mathbf{y}^{q+3})$ and $U(S_{q+3}) = z(\mathbf{x}^{q+3}, \mathbf{y}^{q+3})$.

Step 4 (a): $E_k = \{i \mid i = q+1, q+2, q+3, U(S_i) = -\infty\}$
$P_k = \{i \mid i = q+1, q+2, q+3, \mathbf{x}^i \text{ has all integer components}\}$.

(b): If $P_k \neq \emptyset$, then let

$$z(\mathbf{x}^j, \mathbf{y}^j) = \max \{z(\mathbf{x}^i, \mathbf{y}^i) \mid i \in P_k\}$$

and go to Step 5, otherwise $F_k = G_k = \emptyset$ and go to Step 6.

Step 5: If $z(\mathbf{x}^j, \mathbf{y}^j) \leqq \hat{z}$, then $F_k = P_k$, $G_k = \emptyset$ and go to Step 6.

Otherwise $(\hat{\mathbf{x}}, \hat{\mathbf{y}}) = (\mathbf{x}^j, \mathbf{y}^j)$, $\hat{z} = z(\mathbf{x}^j, \mathbf{y}^j)$, $F_k = P_k$, $k^* = j$.

Form the set

$$G_k = \{i \mid i \in J(k-1) \cup \{q+1, q+2, q+3\} - E_k - P_k, \quad U(S_i) \leqq \hat{z}\}.$$

Step 6: Let

$$J(k) = J(k-1) \cup \{q+1, q+2, q+3\} - E_k - F_k - G_k - \{v(k)\}.$$

if $J(k) = \emptyset$ go to Step 7, otherwise determine

$$U(S_{v(k+1)}) = \max \{U(S_i) \mid i \in J(k)\}.$$

Increase k by 1 and q by 3. If $U(S_{v(k)}) > \hat{z}$ go to Step 2, otherwise go to Step 9.

Step 7: If $k^* = -1$ go to Step 8 otherwise go to Step 9.

Step 8: No solution of (3.36)–(3.40).

Step 9: The optimal solution is $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ with objective function value $\hat{z}$.

The finiteness of this algorithm is obvious. Although set S does not contain a finite number of elements as vector $\mathbf{y}$ contains continuous variables, the sets S_q differ from each other only in the fixed variables of discrete type $\mathbf{x}$. The discrete

variable vector $\mathbf{x}$ may take only a finite number of elements, thus the number of possible S_q sets is finite.

The correctness of the algorithm does not have to be proved, as it is nothing other than the algorithm of Section 3.2 applied to the linear mixed integer problem. The only substantial difference is the use of implicitly bounded sets (explained in detail before the statement of the algorithm).

Note: In Step 2 there are usually several noninteger components of the optimal solution $\mathbf{x}^q$ of the continuous problem. There may be different criteria for choosing one of these (see the next section). A possible choice, for example, being the one the furthest from the next integers, i.e. closest to the nearest half.

Example: (max) $z = 10x_1 + 6x_2 + 8x_3 + 5x_4 + 4y_1 + 3y_2$

subject to

$$12x_1 + 5x_2 + 7x_3 + 10x_4 + 5y_1 + 2y_2 + y_3 = 20$$

$$6x_1 + 11x_2 + 14x_3 + 8x_4 + 3y_1 + 8y_2 + y_4 = 23$$

$$\mathbf{x} \geqq 0, \qquad \mathbf{y} \geqq 0$$

$$x_j \quad \text{integer} \qquad (j = 1, \ldots, 4).$$

Solution: It follows from the constraints that

$$x_1 \leqq 1, \; x_2 \leqq 2, \; x_3 \leqq 1, \; x_4 \leqq 2.$$

The solution of the problem may be followed in Figure 3.3.

Step 1: $J(0) = \emptyset$, $v(1) = 0$, $k = 1$, $\hat{z} = -\infty$, $k^* = -1$.
First of all the problem is solved without the integer restrictions.

$\mathbf{x}^0 = (1.03, 1.53, 0, 0)$, $\mathbf{y}^0 = (0, 0, 0, 0)$, $z_0 = 19.48$, $q = 0$.

Step 2: x_2^0 is chosen, (that is, $\mu(1) = 2$) as a noninteger component because it is the one which lies furthest from the next integers. (Other choice rules may also be used.)
Now the problems

$$\max \{z \mid (\mathbf{x}, \mathbf{y}) \in S_q\}$$

are solved for $q = 1, 2$, where

$$S_1 = \{(\mathbf{x}, \mathbf{y}) \mid (\mathbf{x}, \mathbf{y}) \in S_0, \quad x_2 = 1\}$$

$$S_2 = \{(\mathbf{x}, \mathbf{y}) \mid (\mathbf{x}, \mathbf{y}) \in S_0, \quad x_2 = 2\}$$

and S_0 is the set of vector pairs $(\mathbf{x}, \mathbf{y})$ satisfying the original constraints without the integer restrictions. The set, determined by $x_2 = 0$, is implicitly bounded by set S_1, thus it is not yet considered.

$$\omega(1) = \omega(2) = 0.$$

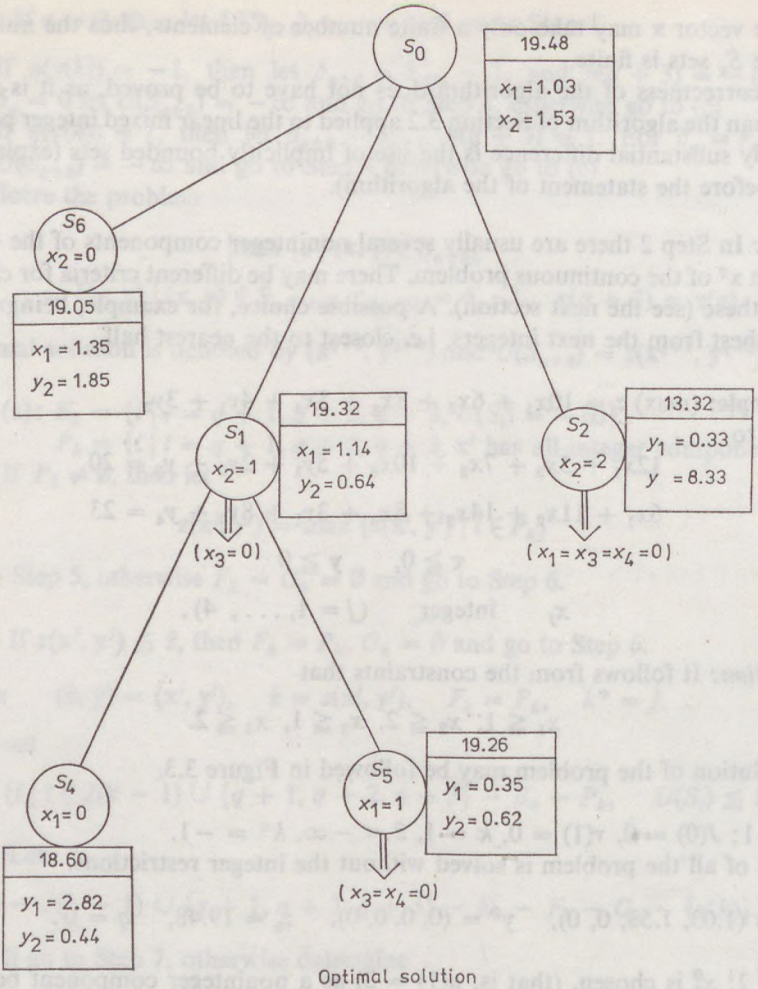


Fig. 3.3

The solution of these problems

$$\mathbf{x}^1 = (1.14, 1, 0, 0) \qquad \mathbf{y}^1 = (0, 0.64, 0, 0) \qquad U(S_1) = 19.32$$

$$\mathbf{x}^2 = (0, 2, 0, 0) \qquad \mathbf{y}^2 = (0.33, 0, 8.33, 0) \qquad U(S_2) = 13.32$$

$$\eta^{(1)} = -1, \qquad \eta^{(2)} = 1$$

as $x_2 = 1$ is less, $x_2 = 2$ is larger than $x_2^0 = 1.53$. $U(S_3) = -\infty$

Step 4 (a): $E_1 = \{3\}$

$P_1 = \{2\}$

(b): P_1 contains only one element, this is the best one as well, thus $j = 2$.

Step 5: $(\hat{\mathbf{x}}, \hat{\mathbf{y}}) = (\mathbf{x}^2, \mathbf{y}^2), \quad \hat{z} = 13.32$

$$F_k = P_k = \{2\}, \quad k^* = 2$$

$$G_k = \{0\}.$$

Step 6: $J(k) = \{0\} \cup \{1, 2, 3\} - \{3\} - \{2\} - \{0\} = \{1\}.$

As $J(k)$ contains only a single element,

$$U(S_{v(2)}) = U(S_1), \quad v(2) = 1, \quad k = 2, \quad q = 3$$

$$U(S_1) > \hat{z} = 13.32.$$

Step 2: There is only a single noninteger component of x^1, $\mu(2) = 1$, $x^1 = 1.14$ x_1 is fixed in S_1 at the values 1 and 2 respectively. In brief,

$$S_4 = \{(x, y) \mid (x, y) \in S_1, \quad x_1 = 1\}, \quad S_5 = \{(x, y) \mid (x, y) \in S_1, \quad x_1 = 2\}.$$

If the corresponding subproblems are solved

$\mathbf{x}^4 = (0, 1, 0, 0)$ $\quad \mathbf{y}^4 = (2.82, 0.44, 0, 0)$ $\quad U(S_4) = 18.60$

$\mathbf{x}^5 = (1, 1, 0, 0)$ $\quad \mathbf{y}^5 = (0.35, 0.62, 0, 0)$ $\quad U(S_5) = 19.26$

$\eta^{(5)} = -1, \quad \eta^{(6)} = +1.$

Step 3: $\eta(v(2)) = \eta(1) = -1$, thus

(a) $\delta_6 = \delta_1 - 1 = 0 \quad \eta(6) = -1$

(c) $S_6 = \{(x, y) \mid (x, y) \in S_0, \quad x_2 = 0\}$

$x^6 = (1.35, 0, 0, 0)$ $\quad y^6 = (0, 1.85, 0, 0)$ $\quad U(S_6) = 19.05.$

Step 4: (a) $E_2 = \emptyset$

$$P_2 = \{4, 5\}$$

$$\max\{z_4, z_5\} = z_5, \quad j = 5.$$

Step 5:

$$z(\mathbf{x}^5, \mathbf{y}^5) > \hat{z}$$

$(\hat{\mathbf{x}}, \hat{\mathbf{y}}) = (\mathbf{x}^5, \mathbf{y}^5)$ $\quad \hat{z} = 19.26, \quad k^* = 5,$

$F_2 = P_2 = \{4, 5\}$ $\quad G_2 = \{6\}.$

Step 6:

$$J(2) = \{1\} \cup \{4, 5, 6\} - \{4, 5\} - \{6\} - \{1\} = \emptyset.$$

Step 7: $k^* \neq -1.$

Step 9: Optimal solution

$$\hat{\mathbf{x}} = (1, 1, 0, 0), \qquad \hat{\mathbf{y}} = (0.35, 0.62, 0, 0)$$

$$\hat{z} = 19.26 \qquad k^* = 5.$$

3.7 An improved branch-and-bound algorithm for mixed integer problems

There are two major drawbacks to the algorithm described in the previous section. First, the upper bounds of the discrete subproblems are obtained by calculating the optimum of the corresponding continuous problem; secondly, the choice of variable to split on may substantially change the direction and thus, the length of the search. The simple rule of choosing the noninteger variable closest to the nearest .5 suggested in the last section does not seem to be a good guide because the change of another variable may be more essential. It is reasonable to make this decision, for example, on the basis of the upper bounds of the objective function obtained after the different fixing of variables. Naturally this is possible only if the upper bounds are calculated very simply.

The description of the method is based on the papers of Tomlin (1969), Beale and Small (1965), and Roy, Benayoun and Tergny (1969). After solving the continuous problem (3.36)–(3.39) of a subproblem of the same structure having stronger lower and upper limit for the discrete variables, the basic variables in the optimal solution are expressed by the nonbasic variables:

$$(\max)\ z = a_{00} + \sum_{j=1}^{k} a_{0j}(-v_j) \tag{3.48}$$

$$u_i = a_{i0} + \sum_{j=1}^{k} a_{ij}(-v_j) \qquad (i = 1, \ldots, m) \tag{3.49}$$

$$\begin{aligned} u_i &\geqq 0 \qquad (i = 1, \ldots, m) \\ v_j &\geqq 0 \qquad (j = 1, \ldots, k). \end{aligned} \tag{3.50}$$

The basic and nonbasic variables are denoted by u_i and v_j respectively; both groups may contain variables of discrete type, $u_i (i \in I)$ and $v_j (j \in J)$, where I and J denote the corresponding sets of subscripts.

The upper and lower bounds – apart from the nonnegativity restrictions – are included in the constraints (3.49). (In practice, they are handled by some kind of upper bounding technique of linear programming.)

Let us denote by $\hat{z}$ the objective function corresponding to the best solution of the original discrete problem (3.36)–(3.40) obtained so far.

As the continuous optimum is supposed as being reached

$$\begin{aligned} a_{0j} &\geqq 0 \qquad (j = 1, \ldots, k) \\ a_{i0} &\geqq 0 \qquad (i = 1, \ldots, m). \end{aligned} \tag{3.51}$$

If $a_{00} \leqq \hat{z}$, the problem (subproblem) is discarded because it may have no better solution than the best one obtained so far. Otherwise, if incidentally all a_{i0} $(i \in I)$ are integers, then the integer problem (subproblem) is solved. In the opposite case let

$$a_{p0} = n_{p0} + f_{p0} \qquad \text{for} \qquad p \in I' \subset I,$$

where n_{p0} is integer, $0 < f_{p0} < 1$, and $I' \neq \emptyset$. Then, as variable u_p must take an integer value,

$$\text{either} \qquad u_p \geqq n_{p0} + 1 \qquad \text{or} \qquad u_p \leqq n_{p0}. \tag{3.52}$$

This determines the branches of solutions. But the set I' usually contains several elements to choose from. On the other hand, even if there is only one fractional a_{i0} $(i \in I)$, the branching results in two new subproblems which are added to the list of subproblems. Thus, the calculation of upper bounds is fully necessary. The quantity which gives a lower bound for the decrease in the objective function if variable u_p is decreased (increased) to the next integer is called the *Down* (*Up*) penalty and denoted by P_D (P_U). According to the theory of linear programming, there is a way to decrease (increase) variable u_p and, at the same time, keep the dual feasibility of the tableau, that is,

$$a_{0j} \geqq 0 \qquad (j = 1, \ldots, m).$$

In both cases, it is easy to determine the nonbasic variable entering the basis, only the minimal ratios

$$\min_{j|a_{pj}>0} a_{0j}/a_{pj} \qquad \text{and} \qquad \min_{j|a_{pj}<0} a_{0j}/(-a_{pj})$$

should be determined. The corresponding changes in the objective function will supply the desired penalties because any dual feasible solution always gives an upper bound of the optimal solution, in maximization problems. Thus

$$P_D = \min_{j|a_{pj}>0} f_{p0}a_{0j}/a_{pj} \tag{3.53}$$

and

$$P_U = \min_{j|a_{pj}<0} (1 - f_{p0})a_{0j}/(-a_{pj}) \tag{3.54}$$

then the new upper bounds of the objective function value if variable $u_p = n_{p0} + 1$ and $u_p = n_{p0}$ are $a_{00} - P_U$ and $a_{00} - P_D$ respectively.

Several consequences may be drawn. Obviously, if

$$\max (a_{00} - P_u, a_{00} - P_D) \leqq \hat{z}, \tag{3.55}$$

then the entire branch may be discarded. If

$$\max (a_{00} - P_U, a_{00} - P_D) > \hat{z} \geqq \min (a_{00} - P_U, a_{00} - P_D), \tag{3.56}$$

then one of the branches may be omitted, preferably the one corresponding to the smaller upper bound. In this case, the present branch is further restricted by adding an inequality for variable u_p.

If

$$\min (a_{00} - P_U, a_{00} - P_D) > \hat{z}^1, \tag{3.57}$$

then both upper bounds are stored and the similar upper bound calculations are maintained for all variables u_i, for which a_{i0} is not integer and $i \in I$.

The algorithmic description of the method could have followed the general framework of Section 3.2 as in Section 3.4, but for certain reasons (to be discussed later), it seems to be more convenient to change the structure to some extent. The selection rules and the decomposition itself are more complicated in the present method, and several steps are needed for this purpose. It will be easy, however, to see the similarities. The notation of the algorithm described in Section 3.2 is completed with the following list.

Notation

T_0	denotes the original problem (3.36)–(3.40) to be solved
T_i	subproblem i, obtained from T_0 by introducing new (stronger) lower and upper bounds for the discrete variables
T_i'	the same as T_i with all variables declared to be continuous
U_i	upper bound of the objective function of problem T'
$(\mathbf{x}^i, \mathbf{y}^i)$	optimal solution of problem T_i'
z^i	optimum in problem T_i'
t	serial number for the subscript of the last problem formed

Now the modified algorithm may be described as follows:

Step 1: Let $J(0) = \emptyset$, $v(1) = 0$, $k = 1$, $\hat{z} = -\infty$, $t = 0$.

Step 2: Solve problem $T'_{v(k)}$.

(a) If there is no solution, then let

$$J(k) = J(k-1) - \{v(k)\}$$

and go to Step 6.

(b) If not all components of $\mathbf{x}^{v(k)}$ are integer, then go to Step 3. In the opposite case, check the inequality

$$z^{v(k)} > \hat{z}.$$

If it is satisfied go to Step 5. Otherwise, let $J(k) = J(k-1) - \{v(k)\}$ and go to Step 6.

Step 3: Denote by I' the set of subscripts i for which $x_i^{v(k)}$ is not integer. Calculate the corresponding penalties

$$P_D(i) \quad \text{and} \quad P_U(i) \quad \text{for} \quad i \in I'.$$

Step 4: Let $D_i = z^{v(k)} - P_D(i)$ and $V_i = z^{v(k)} - P_U(i)$

(a) If there exists a subscript $i \in I'$, for which $\max(D_i, V_i) \leqq \hat{z}$, then let $J(k) = J(k-1) - \{v(k)\}$ and go to Step 6.

(b) If there exist one or more $i \in I'$, for which $\min(D_i, V_i) \leqq \hat{z} < \max(D_i, V_i)$, then the corresponding new restrictions are introduced to problem $T_{v(k)}$

$$x_i \leqq [x_i^{v(k)}] \quad \text{if} \quad D_i > \hat{z} \quad \text{or}$$

$$x_i \geqq [x_i^{v(k)}] + 1 \quad \text{if} \quad V_i > \hat{z}$$

for each i satisfying the inequalities. Then let $v(k+1) = v(k)$, $J(k) = J(k-1)$, k is increased by 1 and go to Step 2.

(c) If neither conditions are satisfied, then choose one of the subscripts in set I', say s (see the discussion after the algorithm), and introduce the following two new problems:

$$T_{t+1} = \{T_{v(k)},\ x_s \leqq [x_s^{v(k)}]\} \qquad U_{t+1} = D_s$$

$$T_{t+2} = \{T_{v(k)},\ x_s \geqq [x_s^{v(k)}] + 1\} \qquad U_{t+2} = V_s.$$

Let

$$J(k) = J(k-1) \cup \{t+1, t+2\} - v(k)$$

$$v(k+1) = \begin{cases} t+1 & \text{if} \quad U_{t+1} \geqq U_{t+2} \\ t+2 & \text{if} \quad U_{t+1} < U_{t+2} \end{cases}$$

increase t by 2 and k by 1. Go to Step 2.

Step 5: A new feasible solution is found. Substitute

$$\hat{z} := z^{v(k)} \quad \text{and} \quad (\hat{\mathbf{x}}, \hat{\mathbf{y}}) := (\mathbf{x}^{v(k)}, \mathbf{y}^{v(k)}).$$

Form the sets

$$G_k = \{i \mid i \in J(k-1),\ U_i \leqq \hat{z}\}$$

$$J(k) = J(k-1) - G_k - \{v(k)\}.$$

Step 6: If $J(k) = \emptyset$ then go to Step 7, otherwise let

$$U_{v(k)} = \max\{U_j \mid j \in J(k)\}$$

and go to Step 2. (See the discussion after the algorithm for other choices of $v(k)$).

Step 7: If $\hat{z} = -\infty$ go to Step 8, otherwise go to Step 9.

Step 8: No feasible solution. Stop.

Step 9: $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ is an optimal solution with an optimal objective function value $\hat{z}$ Stop.

There are two points which need to be discussed in more detail.

Choice of next subproblem for branching

At the first moment, the choice of the branch having maximal upper bound seems to be the best because this is the best branch according to our present knowledge. However, this rule is already broken in the algorithm. Namely, a problem T (with solution set S_i) is divided into problems T_{t+1} and T_{t+2}. Unless both S_{i+1} and S_{i+2}, the corresponding sets of solutions, are empty another problem T_p is not considered for branching even if its upper bound U_p is bigger than U_{t+1} and U_{t+2} (see Step 4). There are several reasons for this. First of all, if the next problem is either T_{t+1} or T_{t+2}, then the reconstruction of the problem (reinversion, reconstruction of other data) is not necessary. Secondly, it is also possible that problem T_p has a better upper bound, because there are only a few discrete variables with integer values and as the process of restricting their values advances the upper bound goes down drastically. Therefore, even if the present branch is closed, the choice of the branch with maximal upper bound (Step 6) may be questionable. Going down in relatively few branches has two advantages. First of all, it is likely to result in feasible solutions earlier in the search. This is safer in the case of long computer runs, so that we do not remain empty-handed after wasting a lot of computer time. At the same time, the better the value $\hat{z}$, the more branches will be omitted in the future in Step 4(a) or 4(b). Secondly, the core of the computer is often a strong limitation. If it is used up by the rapidly increasing number of branches, then the algorithm must be continued as a single branch search until more core is available again. To prevent this awkward situation there are several possibilities. First of all it seems to be reasonable to carry on a single branch search until the upper bound in a different branch is substantionally better than that of the present one. This will solve the problem of jumping too often from one place to another. The question still remains as to how the next branch should be chosen if there are several branches with high upper bounds. It is useful to bring the branches to an equal level as far as the accuracy of upper bounds and limits of discrete variables are concerned. In other words, the cost of restricting further some of the discrete variables should be estimated. This is combined with the upper bound of the corresponding branch.

Choice of the branching variable

After determining the subproblem for decomposition we face the problem of choosing a presently noninteger variable to branch on. The penalties $P_D(i)$ and $P_U(i)$ and the upper bounds

$$D_i = z^{v(k)} - P_D(i) \quad \text{and} \quad V_i = z^{v(k)} - P_U(i) \quad \text{for} \quad i \in I'$$

are determined first. (The definition of I' is in Step 3.) The following three rules will be discussed (s is the subscript of the variable chosen):

(a) Let $W_i = \max(D_i, V_i)$

$W_s = \max\{W_i \mid i \in I'\}$

(b) Let $W_i = |D_i - V_i|$

$W_s = \max\{W_i \mid i \in I'\}$

(c) Let $W_i = \min(D_i, V_i)$

$W_s = \min\{W_i \mid i \in I'\}$.

Rule (a) is the most obvious, though not the best one. Namely, it often happens that on keeping the upper bounds high by choosing rule (a), both D_i and V_i remain high, thus often both alternatives must be explicitly examined. On the other hand, the other variables $i \in I'$ may result in high reductions later, after a great deal of unnecessary calculation. Rule (b) is intended to eliminate this difficulty by choosing the alternatives with the maximal difference and making it likely that one of the alternatives will never have to be examined. Rule (c) is formulated for the same purpose, but from a global point of view. It is called the choice of "worst alternative". In other words, the opposite to the worst alternative is executed in Step 4(c). If, for example,

$$D_s^r \leqq D_i \quad \text{and} \quad D_s \leqq V_i \quad \text{for any} \quad i \in I',$$

then the problem

$$T_{t+2} = \{T_{v(k)}, \quad x_s \geqq [x_s^{v(k)}] + 1\}$$

is chosen for the next branching. Though

$$V_s - D_s < \max_{i \in I'} |V_i - D_i|$$

may happen, thus problem T_{t+1} and T_{t+2} may not be the furthest possible from each other – as far as their objective function value is concerned. But considering the entire procedure, the explicit examination of the unused problem (T_{t+1} in our case) is the least likely in the case of rule (c) because its upper bound estimation is the least one possible.

If the number of discrete variables is considerably high, then it is very time-consuming to calculate the quantities D_i and V_i for each $i \in I'$, all the time. The following ideas may be useful.

(i) Depending on the ranges of the discrete variables or the number of discrete variables already fixed, a greater or lesser number of variable u_i ($i \in I'$) is examined. In other words, an unfortunate decision higher in the solution tree (with more possible descendants) is more harmful; therefore in this case, a bigger proportion of u_i ($i \in I'$) is evaluated.

(ii) It is possible to make a faster but less accurate evaluation of the noninteger variables, and only the best ones – according to this evaluation – are further examined. (In other words, the direct and indirect change of the objective function may be measured first, when a noninteger variable is changed.)

(iii) In the case of pure discrete problems, the effect on the feasibility of the present solution may also be measured.

(iv) It is often useful to divide the discrete variables into subsets according to their meaning in the actual problem and determine a so-called hierarchy graph. In this case a set of discrete variables is examined only if all the discrete variables with higher hierarchy have already integer values.

(v) ε-optimal solution. An approximate solution with an estimation of the optimum is often preferred if a large amount of computing time is expected for the exact solution. In this case only an ε-optimal solution is determined, i.e. a feasible solution being closer to the optimum than ε, where $\varepsilon > 0$, is given in advance. This distance ε is often given in the percentage of the optimum not known in advance. In order to obtain the ε-optimal solution of the problem, only Step 5 of the above algorithm need be modified. In forming set G_k, the inequality $U_i \leqq \hat{z} + \varepsilon$ or $U_i \leqq \hat{z}(1 + \varepsilon)$ will substitute the inequality $U_i \leqq \hat{z}$, depending on whether the maximal distance is given in absolute measure or relative to the optimum.

Stronger penalties

The penalties (3.53) and (3.54) may be improved if we take into account that some of the nonbasic variables are restricted to be integer and if one of them is increased, it must be greater than or equal to 1. If variable v_j is a discrete one, the cost of its increasing to 1 is the reduced cost a_{0j}. If this quantity is larger than the expression corresponding to the subscript j in the equation (3.53) ($a_{pj} > 0$) or in equation (3.54) ($a_{pj} < 0$), then a_{0j} may replace the ratio corresponding to variable v_j. In such a way, as some of the numbers – the minimum of which is taken – are increased, better penalties, say P_D^* and P_U^* are calculated. This calculation may further be improved by introducing the so-called Gomory cuts.

Gomory cuts

Gomory cuts were originally introduced as a means of solving discrete programming problems in the following way. The corresponding continuous problem is solved first. If the continuous optimal solution incidentally satisfies the integer requirements the problem is solved. Otherwise a Gomory cut is introduced which is guaranteed to exclude the present continuous optimal solution, but it does not exclude any feasible solution of the discrete problem. There is a complicated proof for the convergence of the procedure if some simple rules are followed.

This may be found for the pure discrete case for example in Gomory (1963). The mixed integer case is discussed in Gomory (1960). The discussion below on the construction of the Gomory cut is partly based on Hadley (1964).

It is sufficient to consider a single constraint with nonnegative variables

$$u = a_0 + \sum_{j=1}^{k} a_j(-v_j) \tag{3.58}$$

$$u \geqq 0, \quad v_j \geqq 0 \quad (j = 1, \ldots, k)$$

$$u, \quad v_j \quad (j \in J) \text{ integer},$$

which may be one of constraints (3.49) corresponding to a discrete variable u_i, with $a_{i0} > 0$, noninteger, or positive integer combinations of certain such rows in (3.49). As the continuous optimum is obtained by substituting $v_j = 0$ $(j = 1, 2, \ldots, k)$ (see condition (3.51)), a constraint must be determined which excludes this solution but does not exclude any nonnegative solution satisfying the integer requirements.

Such a constraint may be obtained in the following way. Let us introduce the set of subscripts P and N for the positive and negative coefficients in equation (3.58). Furthermore let f_j be the fractional part of coefficient a_j $(j = 0, 1, \ldots, k)$, that is,

$$f_j = a_j - [a_j].$$

Then

$$u = [a_0] + f_0 + \sum_{j=1}^{k} a_j(-v_j).$$

As u must be integer, so must the expression

$$f_0 + \sum_{j=1}^{k} a_j \quad (-v_j)$$

In other words

$$\sum_{j=1}^{k} a_j v_j \equiv f_0 \pmod 1. \tag{3.59}$$

There are two cases one of which is true for any vector $\mathbf{v} \geqq 0$.

Case 1.

$$\sum_{j=1}^{k} a_j v_j \geqq 0.$$

This implies, using congruence (3.59), that

$$\sum_{j=1}^{k} a_j v_j \geqq f_0;$$

thus for the positive coefficients

$$\sum_{j \in P} a_j v_j \geqq f_0. \tag{3.60}$$

Case 2.

$$\sum_{j=1} a_j v_j \leqq 0.$$

Then obviously

$$\sum_{j=1}^{k} a_j v_j \leqq f_0 - 1$$

as a consequence of the above congruence. This means for the negative coefficients that

$$\sum_{j \in N} a_j v_j \leqq f_0 - 1,$$

$$\frac{f_0}{1 - f_0} \sum_{j \in N} (-a_j) v_j \geqq f_0. \tag{3.61}$$

The left hand sides of inequalities (3.60) and (3.61) are always nonnegative because of the nonnegative coefficients and variables. Thus the inequality

$$\sum_{j \in P} a_j v_j + \frac{f_0}{1 - f_0} \sum_{j \in N} (-a_j) v_j \geqq f_0 \tag{3.62}$$

holds unconditionally. If we introduce a new integer variable $s \geqq 0$, a simple form of the Gomory cut may be written as follows:

$$s = -f_0 + \sum_{j \in P} a_j v_j + \frac{f_0}{1 - f_0} \sum_{j \in N} (-a_j) v_j. \tag{3.63}$$

However, if some variables v_j are restricted to be integer, that is, $J \neq \emptyset$, then a further improvement of cut (3.63) is possible.

In congruence (3.59) any expression

$$a_j v_j \quad (j \in J)$$

may be substituted by either

$$f_j v_j \qquad \text{or} \qquad (f_j - 1) v_j$$

because then the sum is changed by an integer. In which case the coefficient in constraint (3.63) becomes

$$f_j \qquad \text{or} \qquad \frac{f_0}{1 - f_0}(1 - f_j).$$

The smaller the coefficient the stronger the constraint; thus, for the final Gomory cut for the mixed integer case, we obtain

$$s = -f_0 + \sum_{j \in J} f_j^* v_j + \sum_{j \in P - J} a_j v_j + \frac{f_0}{1 - f_0} \sum_{j \in N - J} (-a_j) v_j \tag{3.64}$$

where

$$f_j^* = \min\left\{f_j, \frac{f_0}{1-f_0}(1-f_j)\right\} = \begin{cases} f_j & \text{if} \quad f_j \leqq f_0 \\ \dfrac{f_0}{1-f_0}(1-f_j) & \text{if } f_j > f_0 . \end{cases} \tag{3.64}$$

If we write this inequality in the form

$$s = -f_0 - \sum_{j=1} f_j^*(-v_j) \tag{3.65}$$

where $f_j^* \geqq 0$, then

$$P_G = \min_j f_0 a_{0j}/f_j^*$$

is obviously a penalty. It is easy to see, that

$$P_G \geqq \min\,(P_u^*, P_D^*)$$

thus a stronger rule for omitting the present subproblem may be obtained, viz:

$$P_G \geqq a_{00} - \hat{z}.$$

CHAPTER 4

DYNAMIC PROGRAMMING AS A TOOL FOR SOLVING DISCRETE PROGRAMMING PROBLEMS

4.1 Introduction

The theory of dynamic programming, designed to solve multi-stage decision problems was developed by Richard Bellman (1957). It is closely related to several branches of mathematics: functional equations, differential and integral equations, calculus of variations etc.

In the present work only a very simple form of the theory of dynamic programming will be used for solving discrete programming problems of several variables by decomposition. In other words, a series of one-dimensional problems will be solved instead. It should be noted however, that different problems of dynamic programming may be reformulated as discrete programming problems and may be solved by other methods of discrete programming.

4.2 Optimality principle of dynamic programming

The optimality principle will be presented using the following simple economic multi-decision problem.

A sum e is given for purchasing machines of type A and B for the unit price a and b respectively. At the end of the year they are sold for the prices $c < a$ and $d < b$ respectively. The quantity of production in a year is denoted by $g(u)$ and $h(v)$, where u and v are the respective numbers of purchased machines of type A and B. A natural measure (number of items, weight, etc.) may be used for the production. The money obtained for the machines at the end of the year is used to buy new ones under the same conditions. The process is continued for n years and an optimal policy is to be determined, i.e. the policy of optimizing the total production of n years.

More realistic models can be formed by adding new constraints to the problem without substantially changing the procedure. In order to keep the discussion as clear as possible, the above original problem will be considered.

Denoting by u_i and v_i the respective machines of type A and B purchased at the beginning of year i, the problem may be formulated as follows

$$\max \sum_{i=1}^{n} \left(g(u_i) + h(v_i)\right) \tag{4.1}$$

$$au_1 + bv_1 \leqq e \tag{4.2}$$

$$au_i + bv_i \leqq cu_{i-1} + dv_{i-1} \qquad (i = 2, \ldots, n) \tag{4.3}$$

$$u_i \geqq 0, \quad v_i \geqq 0 \qquad (i = 1, 2, \ldots, n) \tag{4.4}$$

$$u_i, v_i \text{ integer} \qquad (i = 1, 2, \ldots, n) \tag{4.5}$$

where a, b, c, d, e are given positive integers, for which $c < a$ and $d < b$, furthermore g and h are arbitrary functions defined on the nonnegative integers. This problem always has an optimal solution. On the one hand the set of feasible solutions defined by the constraints (4.2)–(4.5) is never empty because $u_i = v_i = 0$ $(i = 1, 2, \ldots, n)$ is always a solution. On the other hand this set is also bounded. This is obvious from constraints (4.2) and (4.4) for variables u_1 and v_1, and it follows for the rest of the variables by induction.

The problem will be reformulated with the help of the optimality principle of dynamic programming stated first by Bellman.

Optimality principle

An optimal decision policy always has the property that whatever the initial state of the system and the initial decision are, the rest of the decisions must form an optimal policy according to the state produced by the first decision.†

The proof of this principle is obvious because the opposite statement immediately results in a contradiction. Behind this principle there is the simple mathematical fact that the maximum of a function is never smaller on a set than on any of its subsets. In the present case the feasible decision policies form the set and the subset is determined by fixing the first decision.

Now (4.1)–(4.5) may be reformulated using this principle. Let us denote by x_k the money available for the year k. $T(x)$ is the set of decisions (u, v) permitted by a sum x.

$$x_k = cu_{k-1} + dv_{k-1} \qquad (k = 2, \ldots, n) \tag{4.6}$$

$$T(x) = \{(u, v) \mid u \geqq 0, \quad v \geqq 0 \quad \text{integer}, \quad au + bv \leqq x\}. \tag{4.7}$$

Problem (4.1)–(4.5) is now embedded in a set of problems:

$$\max \sum_{i=1}^{k} (g(u_i) + h(v_i)) \tag{4.8}$$

$$au_1 + bv_1 \leqq x \tag{4.9}$$

$$au_i + bv_i \leqq x_i \qquad (i = 2, \ldots, k) \tag{4.10}$$

$$u_i \geqq 0, \quad v_i \geqq 0 \quad \text{integer} \qquad (i = 1, 2, \ldots, k) \tag{4.11}$$

$$x_i = cu_{i-1} + dv_{i-1} \qquad (i = 2, \ldots, k). \tag{4.12}$$

† In this principle it is indirectly supposed that the later decisions do not change the value of the previous ones. This note will become obvious when the recursion of dynamic programming is discussed.

The optimum of problem (4.8)–(4.12) is denoted by

$$f_k(x) \qquad (0 \leqq x \leqq e, \quad k = 1, 2, \ldots, n).$$

As our decisions are represented by variables (u_i, v_i), they are called decision variables or decision parameters. On the other hand the state of the system may be characterized by the amount of money x_i available for the next period, they are therefore called *state variables* or *state parameters.*

Function $f_1(x)$ can easily be determined for all $0 \leqq x \leqq e$ from the definition:

$$f_1(x) = \max \{g(u) + h(v) \mid (u, v) \in T(x)\}. \tag{4.13}$$

To determine the function $f_2(x)$, the principle of optimality is used. No matter what the first decision (u, v) is, the continuation must be optimal, according to the sum of money $cu_1 + dv_1$, available for the second period. The best solution for the two-period problem therefore can be found by comparing the total returns resulting from the different decisions (u, v) and from their optimal continuations

$$f_2(x) = \max \{g(u) + h(v) + f_1(cu + dv) \mid (u, v) \in T(x)\} \tag{4.14}$$

for all $0 \leqq x \leqq e$. Similarly, after determining $f_{k-1}(x)$, in the same way

$$f_k(x) = \max \{g(u) + h(v) + f_{k-1}(cu + dv) \mid (u, v) \in T(x)\} \tag{4.15}$$

for all $0 \leqq x \leqq e$ $(k = 2, 3, \ldots, n)$. We can suppose that the numbers a, b, c, d, e are positive integer. Then the maximizations (4.13) and (4.15), the latter for $k = 2, \ldots, n$, mean a finite number of comparisons. When determining the value $f_k(x)$, the corresponding best pair (u, v) is also recorded and denoted by $(\hat{u}^k(x), \hat{v}^k(x))$ for all $k = 1, 2, \ldots, n$ and $x = 0, 1, \ldots, e$. The optimum of the original problem (4.1)–(4.5) is $f_n(e)$. Then the optimal solution of the problem $(\hat{u}_k, \hat{v}_k)$ $(k = 1, 2, \ldots, n)$ may easily be determined in the following way.

Obviously

$$(\hat{u}_1, \hat{v}_1) = (u^n(e), v^n(e)).$$

by the definition of functions $f_n(x)$ and $u^n(x)$, $v^n(x)$. Then the sum of money available for the $n - 1$ period problem is

$$x_2 = c\hat{u}_1 + d\hat{v}_1$$

thus the first decision (which is the second decision of the original problem) is

$$(\hat{u}_2, \hat{v}_2) = (u^{n-1}(x_2), v^{n-1}(x_2)).$$

similarly,

$$(\hat{u}_k, \hat{v}_k) = (u^{n-k+1}(x_k), v^{n-k+1}(x_k))$$

where

$$x_k = c\hat{u}_{k-1} + d\hat{v}_{k-1} \qquad (k = 2, 3, \ldots, n).$$

Note: Obviously it is sufficient to determine function $f_k(x)$ of the interval

$$0 \leqq x \leqq e\left\{\max\left(\frac{c}{a}, \frac{d}{b}\right)\right\}^{n-k}.$$

If these intervals are considered, then the procedure may also be applied in the case $c \geqq a$ and/or $d \geqq b$.

Then the multiplier of e is greater than or equal to 1, that is, the interval to be considered is nondecreasing with k.

4.3 Solution of the knapsack problem by dynamic programming

The knapsack problem was formulated in Chapter 1, and solved in Chapter 3. Now it is extended to the non 0–1 integer problem

$$\max \sum_{j=1}^{n} c_j x_j \tag{4.16}$$

$$\sum_{j=1}^{n} a_j x_j \leqq K \tag{4.17}$$

$$x_j \in \{0, 1, \ldots, r_j\} \qquad (j = 1, \ldots, n) \tag{4.18}$$

where $r_j \geqq 1$, a_j, c_j, K are given positive integers. This problem may also be considered as a multi-stage decision problem. The decision variables are $x_1, x_2, \ldots, x_n$ and only one at a time is determined. The parameter describing the "state" of the problem is nothing but the capacity still available, i.e. it is K at the beginning and for example $K - a_1x_1$, after the first decision. Now (4.16)–(4.18) is embedded in a family of problems:

$$\max \sum_{j=1}^{k} c_j x_j \tag{4.19}$$

$$\sum_{j=1}^{k} a_j x_j \leqq y \tag{4.20}$$

$$x_j \in \{0, 1, \ldots, r_j\} \qquad (j = 1, \ldots, k). \tag{4.21}$$

This problem is solved for all $k = 1, 2 \ldots, n$ and $y = 0, 1, \ldots, k$, and its optimum is denoted by $f_k(y)$. Obviously,

$$f_1(y) = \max \{c_1x_1 \mid x_1 \in \{0, 1, \ldots, r_1\};\quad a_1x_1 \leqq y\}. \tag{4.22}$$

In solving the two variable problems one must be careful because the stages are not equivalent – unlike the previous problem. Thus the first decision must be the value of the second variable because in such a way a one-variable problem

remains which contains the first variable. The optimality principle may be used, i.e. after the first decision the remaining capacity must be used in an optimal way:

$$f_2(y) = \max \{c_2x_2 + f_1(y - a_2x_2) \mid x_2 \in \{0, 1, \ldots, r_2\}, a_2x_2 \leqq y\}.$$

Similarly after determining the function $f_{k-1}(y)$, the optimum of the k variable problem may be determined by the expression

$$f_k(y) = \max \{c_kx_k + f_{k-1}(y - a_kx_k) \mid x_k \in \{0, 1, \ldots, r_k\}, a_kx_k \leqq y\}. \quad (4.23)$$

These functions ($k = 2, \ldots, n$) are determined for the arguments $y = 0, 1, \ldots, K$.

Let us denote by $x_k^*(y)$ the value of the decision variable for which the maximum is reached in determining the function $f_k(x)$. Then the optimum of problem (4.16)–(4.18) is obviously $f_n(K)$. The corresponding optimal solution ($\hat{\mathbf{x}}$) is easily determined. Obviously

$$\hat{x}_n = x_n^*(K)$$

by the definition of functions $f_n(y)$ and $x_n^*(y)$. The remaining capacity for the $n - 1$ variable problem is $K - a_n\hat{x}_n$, thus

$$\hat{x}_{n-1} = x_{n-1}^*(K - a_n\hat{x}_n)$$

and similarly

$$\hat{x}_k = x_k^*\left(K - \sum_{j=k+1}^{n} a_j\hat{x}_j\right) \qquad (k = n - 1, n - 2, \ldots, 1).$$

Example: Solve the following problem

$$\max 8x_1 + 5x_2 + 3x_3$$

$$4x_1 + 3x_2 + 2x_3 \leqq 14$$

$$x_1, x_2, x_3 \geqq 0 \text{ integer.}$$

First of all the functions f_k ($k = 1, 2, 3$) are determined for $y = 0, 1, \ldots, 14$. They are recorded in tableau format (Table 4.1). As some of the rows are obviously identical, they are drawn together for compactness.

TABLE 4.1

y	$x_1^*(y)$	$f_1(y)$
0–3	0	0
4–7	1	8
8–11	2	16
12–14	3	24

The function $f_1(x)$ may be determined explicitly

$$x_1^*(y) = \min\left(r_1, \left[\frac{y_1}{a_1}\right]\right), \quad f_1(y) = c_1 x_1^*(y)$$

where $a_1 = 4$, $c_1 = 8$, and r_1 is not given (considered as ∞). In tableau format, this is illustrated in Table 4.1.
Function $f_2(y)$ may easily be calculated from the definition

$$f_2(y) = \max\{5x_2 + f_1(y - 3x_2) \mid x_2 \geqq 0 \text{ integer}; \quad 3x_2 \leqq y\}$$

as there are only a small number of choices. Function $f_2(y)$ is the following (Table 4.2):

TABLE 4.2

y	$x_2^*(y)$	$f_2(y)$
0–2	0	0
3	1	5
4–5	0	8
6	2	10
7	1	13
8–9	0	16
10	2	18
11	1	21
12–13	0	24
14	2	26

To solve the given numerical problem only $f_3(14)$ and $x_3^*(14)$ are to be calculated

$$f_3(14) = \max\{3x_3 + f_2(14 - 2x_3) \mid x_3 \in \{0, 1, \ldots, 7\}\}$$

thus

$$x_3^*(14) = 1, \qquad f_3(14) = 27$$

which is the optimum of the problem. The optimal solution may easily be determined

$$\hat{x}_3 = x_3^*(14) = 1$$

$$\hat{x}_2 = x_2^*(14 - 2) = 0$$

$$\hat{x}_1 = x_1^*(14 - 2 - 0) = 3,$$

that is

$$\hat{\mathbf{x}} = (3, 0, 1).$$

A little additional work – calculation of the tableau $f_n(y)$, $x_n^*(y)$ – provides a parametric solution of the problem for any right hand side in the interval $[0, K]$.

4.4 Solution of discrete problems of several constraints

For the sake of simplicity, let us consider a problem of two constraints, noting that the calculation for the several constraint case can be carried on similarly. The cargo loading problem of Chapter 1 is the following:

$$\max \sum_{j=1}^{n} c_j x_j \tag{4.24}$$

$$\sum_{j=1}^{n} a_j x_j \leqq K \tag{4.25}$$

$$\sum_{j=1}^{n} b_j x_j \leqq L \tag{4.26}$$

$$x_j \in \{0, 1, \ldots, r_j\} \qquad (j = 1, 2, \ldots, n) \tag{4.27}$$

where $r_j \geqq 1$, $a_j, b_j \geqq 0$ $(a_j + b_j > 0)$, $c_j, K, L > 0$ are given integers. Similarly to the method used in the previous section, (4.24)–(4.27) is embedded in a family of problems

$$\max \sum_{j=1}^{k} c_j x_j \tag{4.28}$$

$$\sum_{j=1}^{k} a_j x_j \leqq y \tag{4.29}$$

$$\sum_{j=1}^{k} b_j x_j \leqq w \tag{4.30}$$

$$x_j \in \{0, 1, \ldots, r_j\} \qquad (j = 1, 2, \ldots, k). \tag{4.31}$$

The optimal value of the objective function of this problem is denoted by

$$f_k(y, w)$$

and it is calculated for

$$k = 1, 2, \ldots, n.$$
$$y = 0, 1, \ldots, K$$
$$w = 0, 1, \ldots, L.$$

The first of these functions may be calculated by the definition

$$f_1(y, w) = \max \{c_1 x_1 \mid x_1 \in \{0, 1, \ldots, r_1\}, a_1 x_1 \leqq y, \quad b_1 x_1 \leqq w\} \tag{4.32}$$

for $y = 0, 1, \ldots, K$; $w = 0, 1, \ldots, L$ independently from each other. To determine the second function the principle of optimality is used. For the same reason as before, the second variable is fixed at a permitted value and the continuation must

be optimal – according to the remaining capacity. The best choice of variable x_2 results in the function f_2

$$f_2(y, w) = \max \{c_2x_2 + f_1(y - a_2x_2, w - b_2x_2) \mid x_2 \in \{0, 1, \ldots, r_2\}, a_2x_2 \leqq y, b_2x_2 \leqq w\}$$

where $y = 0, 1, \ldots, K$ and $w = 0, 1, \ldots, L$ independently from each other. Similarly if function f_{k-1} is already determined, then

$$f_k(y, w) = \max \{c_kx_k + f_{k-1}(y - a_kx_k, w - b_k\widehat{x_k}) \mid x_k \in \{0, 1, \ldots, r_k\}; a_kx_k \leqq y, b_kx_k \leqq w\} \tag{4.33}$$

where $y = 0, 1, \ldots, K$ and $w = 0, 1, \ldots, L$ independently from each other.

When determining function $f_k(y, w)$ $(k = 1, \ldots, n)$, the value of variable x_k, for which the maximum (4.32) and (4.33), respectively, is attained, will be denoted by

$$x_k^*(y, w).$$

Now the optimal solution of (4.24)–(4.27) which is denoted by $\hat{\mathbf{x}}$, is

$$\hat{x}_k = x_k^*(y_k, w_k) \qquad (k = n, n-1, \ldots, 1)$$

where

$$y_n = K, \; w_n = L$$

$$y_k = y_{k+1} - a_{k+1}\hat{x}_{k+1} \qquad (k = n-1, \ldots, 1) \tag{4.34}$$

$$w_k = w_{k+1} - b_{k+1}\hat{x}_{k+1}. \qquad (k = n-1, \ldots, 1)$$

Example:

$$\max 11x_1 + 30x_2 + 17x_3 + 25x_4$$

$$x_1 + 2x_2 + x_3 + 2x_4 \leqq 5$$

$$2x_1 + 3x_2 + 3x_3 + x_4 \leqq 6$$

$$x_j \geqq 0, \quad \text{integer} \qquad (j = 1, 2, 3, 4).$$

It follows from the constraints that the functions have to be determined only for

$$x_1 \leqq 3, \quad x_2 \leqq 2, \quad x_3 \leqq 2, \quad x_4 \leqq 2.$$

It is obvious that

$$x_k^*(0, w) = x_k^*(y, 0) = 0 \qquad (k = 1, \ldots, n)$$

hence

$$f_k(0, w_k) = f_k(y_k, 0) = 0 \qquad (k = 1, \ldots, n)$$

therefore these values will not be indicated in the function tableaux (Tables 4.3 and 4.4). The first function, f_1 may be determined explicitly

$$x_1^*(y, w) = \min \left\{ \left[\frac{y}{a_1} \right], \left[\frac{w}{b_1} \right] \right\}^{\dagger}$$

† If one of the denominators is 0, then the corresponding term is considered as ∞. Both a_1 and b_1 cannot be zero according to the supposition $a_1 + b_1 > 0$.

and

$$f_1(y, w) = c_1 x_1^*(y, w).$$

Functions f_2 and f_3 may be calculated by recursion (4.33). Only a small number of substitutions and comparisons should be carried out for each function value:

TABLE 4.3

$f_2(y, w)$

w \ y	1		2		3		4		5	
	f_2	x_2^*	f_2	x_2^*	f_2	x_2^*	f_2	x_2^*	f_2	x_2^*
1	0	0	0	0	0	0	0	0	0	0
2	11	0	11	0	11	0	11	0	11	0
3	11	0	30	1	30	1	30	1	30	1
4	11	0	30	1	30	1	30	1	30	1
5	11	0	30	1	41	1	41	1	41	1
6	11	0	30	1	41	1	60	2	60	2

TABLE 4.4

$f_3(y, w)$

w \ y	1		2		3		4		5	
	f_3	x_3^*	f_3	x_3^*	f_3	x_3^*	f_3	x_3^*	f_3	x_3^*
1	0	0	0	0	0	0	0	0	0	0
2	11	0	11	0	11	0	11	0	11	0
3	17	1	30	0	30	0	30	0	30	0
4	17	1	30	0	30	0	30	0	30	0
5	17	1	30	0	41	0	41	0	41	0
6	17	1	34	2	47	1	60	0	60	0

If only the original numerical example is to be solved, then

$$f_4(5, 6) = \max [f_3(5, 6), \quad 25 + f_3(3, 5), \quad 50 + f_3(1, 4)\} = 67$$

and

$$\hat{x}_4 = x_4^*(5, 6) = 2$$

$$y_3 = 1, \; w_3 = 4$$

$$\hat{x}_3 = x_3^*(1, 4) = 1$$

as

$$y_2 = 0,$$

$$\hat{x}_2 = \hat{x}_1 = 0$$

thus the optimal solution of the problem

$$\hat{\mathbf{x}} = (0, 0, 1, 2)$$

and the optimal objective function value, $f_4(5, 6) = 67$. The parametric solution of the problem for changing right hand sides, would only require the calculation of functions $f_4(y, w)$ and $x_4^*(y, w)$ for all values $y = 0, 1, \ldots, 5$ and $w = 0, 1, \ldots, 6$.

Note 1: The method is designed for computers. It is not necessary to keep all function tableaux in the core. When calculating functions f_k, x_k^*, only function f_{k-1} is needed. After obtaining all the functions, when the optimal solution is calculated only one of the functions x_k^* at a time must be in the core.

Note 2: It is easy to see that the amount of computation grows linearly with the number of variables and exponentially with the number of constraints (number of state parameters). This means that problems of many variables but of few constraints can be handled effectively by dynamic programming. In the following sections, the method is improved for the general case, i.e. for problems with many constraints.

Note 3: Separable nonlinear problems can be solved effectively by dynamic programming if the number of constraints is low or the space of state variables $(y, w, \ldots)$ may otherwise be limited.

4.5 Reducing the number of optimum functions

In order to reduce the amount of computation in the dynamic optimization methods let us examine the algorithm presented in the foregoing section, including the embedded process itself.

A few years ago it was discovered that it was unnecessary to calculate several functions of optimum for different numbers of variables (See, e.g. Shapiro and Wagner (1967) and Greenberg (1969a)).

To illustrate the idea let us consider the problem of the last section, which is repeated here for convenience:

$$\max \sum_{j=1}^{n} c_j x_j \tag{4.35}$$

$$\sum_{j=1}^{n} a_j x_j \leqq K \tag{4.36}$$

$$\sum_{j=1}^{n} b_j x_j \leqq L \tag{4.37}$$

$$x_j \geqq 0 \text{ integer} \qquad (j = 1, 2, \ldots, n) \tag{4.38}$$

where $a_j, b_j \geqq 0$ $(a_j + b_j > 0)$, $c_j, K, L > 0$ are given integers. The problem is now embedded in a set of problems

$$\max \sum_{j=1}^{n} c_j x_j \tag{4.39}$$

$$\sum_{j=1}^{n} a_j x_j \leqq y \tag{4.40}$$

$$\sum_{j=1}^{n} b_j x_j \leqq w \tag{4.41}$$

$$x_i \geqq 0 \text{ integer} \qquad (j = 1, 2, \ldots, n). \tag{4.42}$$

In contrast to the last section all problems in the family have n variables like the original problem. It should be noted however, that the variables may not be bounded explicitly.

The optimum of (4.39)–(4.42) is denoted by

$$f(y, w)$$

and it is calculated for $y = 0, 1, \ldots, K$ and $w = 0, 1, \ldots, L$.

The recursion for calculating the function $f(y, w)$ and for determining the optimal solution $\hat{\mathbf{x}}$ is however entirely different. The substance of dynamic programming is the decomposition of one compound decision to several consecutive simple ones. In our problem (4.35)–(4.38), the composite decision is the optimal solution $\hat{\mathbf{x}}$. In the algorithm of the foregoing sections one variable was determined at a time. In the new approach of the present section one decision consists of increasing one variable by only one unit. If variable j is chosen, the corresponding gain is c_j. According to the principle of optimality (Section 4.2), the remaining capacities have to be used in an optimal way. In order to obtain the optimum for capacities (y, w) the sum of immediate return (c_j) and the optimum corresponding to the remaining capacities $(y - a_j, w - b_j)$ should be calculated, that is,

$$f(y, w) = \max \{c_j + f(y - a_j, w - b_j) \mid j \in 1, \ldots, n\}, \quad a_j \leqq y, \quad b_j \leqq w\}. \tag{4.43}$$

The subscript j for which the maximum is attained in the right hand side of equation (4.43) is denoted by

$$j^*(y, w).$$

This subscript function will play the role of the function $x_j^*(y, w)$ of the last section. If this subscript is not uniquely determined by the maximization, then one of them (e.g. the smallest) is chosen. In other words, instead of n functions of the optimum, only one function is to be determined.

The function $f(y, w)$ may be determined easily from the recursion (4.43) for increasing y and w starting from

$$f(0, 0) = 0$$

and $$f(y, w) = 0 \qquad \text{if} \quad y < \min_j a_j \quad \text{or} \quad w < \min_j b_j.$$

The corresponding optimal subscript by definition

$$j^*(y, w) = 0.$$

After obtaining the value $f(K, L)$ and the corresponding subscript $j^*(K, L)$ the optimal solution $\hat{\mathbf{x}}$ of (4.35)–(4.38) can be obtained. Let us determine the following series of subscripts

$$j_1 = j^*(K, L) \tag{4.44}$$

$$j_k = j^*\left(K - \sum_{r=1}^{k-1} a_{j_r},\ L - \sum_{r=1}^{k-1} b_{j_r}\right) \quad k = 2, 3, \ldots, N$$

where $j_{N-1} > 0$ and $j_N = 0$.

This is a finite series in that it was supposed that $a_j + b_j > 0$; thus, with each new subscript determined either the first or the second capacity decreases.

The optimal solution $\hat{\mathbf{x}}$ can be calculated easily:

$$\hat{x}_j = \sum_{r=1}^{N} \delta(j, j_r) \qquad (j = 1, 2, \ldots, n)$$

where

$$\delta(j, j_r) = \begin{cases} 1, & \text{if } j = j_r \\ 0, & \text{otherwise,} \end{cases}$$

that is, by counting the identical subscripts in the series (4.44).

The example of Section 4.4 is resolved as an illustration.

Example:

$$\max 11x_1 = 30x_2 + 17x_3 + 25x_4$$

$$x_1 + 2x_2 + x_3 + 2x_4 \leqq 5$$

$$2x_1 + 3x_2 + 3x_3 + x_4 \leqq 6$$

$$x_j \geqq 0, \text{ integer} \qquad (j = 1, 2, 3, 4).$$

Obviously

$$f(y, w) = 0 \quad \text{if} \quad y = 0 \quad \text{or} \quad w = 0$$

and in these cases

$$j^*(y, w) = 0$$

by definition. Therefore the tableau of functions f and j^* will be calculated for $y \geqq 1$ and $w \geqq 1$ in an increasing order of arguments, for example row-wise. In other words these functions are calculated for $w = 1$, $y = 1, 2, \ldots, 5$, then for $w = 2$, $y = 1, 2, \ldots, 5$, etc. For example, if the functions f and j^* are known for

$$y \leqq 3, \quad w < 4 \quad \text{and} \quad y \leqq 3, \quad w \leqq 4,$$

then

$$f(3, 4) = \max\{c_1 + f(2, 2), \quad c_2 + f(1, 1), \quad c_3 + f(2, 1), \quad c_4 + f(1, 3)\} =$$

$$= \max\{11 + 25, \quad 30 + 0, \quad 17 + 25, \quad 25 + 17\} = 42$$

$$j^*(3, 4) = 3,$$

because the maximum is attained for $j = 3$ and 4 and the smaller subscript is accepted. The whole of Table 4.5 may be obtained similarly.

TABLE 4.5

w \ y	1		2		3		4		5	
	f	j^*	f	j^*	f	j^*	f	j^*	f	j^*
1	0	0	25	4	25	4	25	4	25	4
2	11	1	25	4	25	4	50	4	50	4
3	17	2	30	2	36	1	50	4	50	4
4	17	2	30	2	42	3	55	2	61	1
5	17	2	30	2	42	3	55	2	67	3
6	17	2	34	3	47	2	60	2	67	3

Thus the optimum of the problem

$$f(5, 6) = 67.$$

The optimal solution may be obtained by calculating the subscripts (4.44)

$$j_1 = j^*(5, 6) = 3$$

$$j_2 = j^*(4, 2) = 4$$

$$j_3 = j^*(2, 1) = 4$$

$$j_4 = j^*(0, 0) = 0$$

and counting the subscripts

$$\hat{\mathbf{x}} = (0, 0, 1, 2).$$

Notes: The algorithm presented here is more efficient than that of Section (4.4). The amount of computation is smaller by a factor of n. On the other hand – as already mentioned – this more compact form does not allow explicit upper bounds of the variables. This is a consequence of the fact that the variables themselves are not recorded during the calculation of the optimum function f, thus it is not known when the upper bound is reached. Other methods, avoiding this restriction, will be discussed later.

4.6 Extension of the single optimum function method to the bounded variable case

This approach originates from the work of Greenberg (1969) although some changes are introduced. Let us consider the problem in an equation form. (If the problem contains inequality restrictions, they are transformed into equations by adding integer slack or surplus variables.)

$$\min \mathbf{c}^T\mathbf{x} \tag{4.45}$$

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{4.46}$$

$$0 \leqq x_j \leqq r_j \qquad (j = 1, \ldots, n) \tag{4.47}$$

$$x_j \text{ integer} \qquad (j = 1, \ldots, n). \tag{4.48}$$

In contrast to (4.24)–(4.27) the nonnegativity of the coefficients **A**, **c** and the right hand side **b** is not supposed. $r_j > 0$ $(j = 1, \ldots, n)$ are given integers.

The continuous problem (4.45)–(4.47) is solved first using the upper-bounding technique of Dantzig (1963). In this algorithm the basis corresponds only to the constraints (4.46) and the upper bounds are taken into account another way. Let us denote the optimal basis by **B** and let us decompose the matrix **A** and vector **x** accordingly:

$$\mathbf{A} = [\mathbf{B}, \mathbf{R}], \quad \mathbf{x} = \begin{bmatrix} \mathbf{x}_B \\ \mathbf{x}_R \end{bmatrix}, \quad \mathbf{c}^T = [\mathbf{c}_B^T, \mathbf{c}_R^T] \quad J = \{1, 2, \ldots, n\} = J_B \cup J_R,$$

then the problem becomes

$$\min \mathbf{c}_B^T \mathbf{x}_B + \mathbf{c}_R^T \mathbf{x}_R \tag{4.49}$$

$$\mathbf{B}\mathbf{x}_B + \mathbf{R}\mathbf{x}_R = \mathbf{b} \tag{4.50}$$

$$0 \leqq x_j \leqq r_j, \quad x_j \text{ integer} \qquad (j = 1, \ldots, n) \tag{4.51}$$

or expressing the vector $\mathbf{x}_B$ from equation (4.50) and substituting it in the objective function:

$$\min \tilde{\mathbf{c}}_R^T \mathbf{x}_R \tag{4.52}$$

$$\mathbf{x}_B = \mathbf{B}^{-1}\mathbf{b} - \mathbf{B}^{-1}\mathbf{R}\mathbf{x}_R \tag{4.53}$$

$$\mathbf{x}_R, \mathbf{x}_B \text{ integer} \tag{4.54}$$

$$0 \leqq x_j \leqq r_j \quad j \in J_R \tag{4.55}$$

$$0 \leqq x_j \leqq r_j \quad j \in J_B \tag{4.56}$$

where

$$\tilde{\mathbf{c}}_R = \mathbf{c}_R - \mathbf{c}_B\mathbf{B}^{-1}\mathbf{R} \geqq \mathbf{0}.$$

The nonnegativity of vector $\tilde{\mathbf{c}}_R$ is a consequence of the optimality of basis **B**. If $\mathbf{x}_B = \mathbf{B}^{-1}\mathbf{b}$ has only integer components by chance, then no further calculation is necessary. But usually the opposite is true in which case the calculation is continued in a second phase.

In the cutting plane method of Gomory (1963) a series of continuous problems is solved until the last problem has an integer optimal solution. The opposite is done in the present approach. A series of problems of type (4.52)–(4.55) is solved (possibly with larger than 0 lower bounds). The feasible solutions of (4.52)–(4.55) are generated in an order of nondecreasing costs until the constraint (4.56) is also satisfied. Problem (4.52)–(4.55) is reformulated and embedded in the following set of problems

$$\min \sum_{j \in J_R} c_j' x_j \tag{4.57}$$

$$\sum_{j \in J_R} \beta_j x_j = \beta \pmod{D} \tag{4.58}$$

$$0 \leqq x_j \leqq r_j \quad x_j \quad \text{integer} \quad j \in J_R \tag{4.59}$$

where β is a parameter vector of integer components and

$$c'_j = \tilde{c}_j D, \ \beta_j = \mathbf{B}^{-1}\mathbf{a}_j D, \ \beta_0 = \mathbf{B}^{-1}\mathbf{b} D, \ D = |\mathbf{B}|$$

where $\mathbf{a}_j$ is the jth column vector of matrix $\mathbf{R}$.

Problem (4.52)–(4.55) is equivalent to (4.57)–(4.59) with right hand side $\beta = \beta_0$. D might be the LCM (least common multiple) of the denominators in the components of vectors $\mathbf{B}^{-1}\mathbf{a}_j$ and $\mathbf{B}^{-1}\mathbf{b}$. We are using $D = \det \mathbf{B}$ for simplicity. Let us denote the optimum of (4.57)–(4.59) by $F(\beta)$. $j^*(\beta)$ will denote the highest subscript of nonzero variables in the corresponding optimal solution. It is easy to formulate the dynamic programming recursion for function $F(\beta)$:

$$F(\beta) = \min_{j \in J_R} [c'_j + F(\beta - \beta_j)]. \tag{4.60}$$

The arguments of function F are taken mod D.

Note 1°: If all optimal solutions of the problem with right hand side $\beta - \beta_j$ contain $x_j = r_j$, then subscript j should be excluded when determining the value $F(\beta)$.

Note 2°: In order to avoid duplication of solutions the components of which are obtained in different order, the following rule will be applied: *Only chains of nondecreasing variable subscripts are accepted*, that is

$$j_1 \leqq j_2 \leqq \cdots \leqq j_k \leqq \cdots . \tag{4.61}$$

In other words a solution may be extended only by increasing the positive element of highest subscript or any variable with even higher subscript.

Now we shall construct an algorithm to determine the values $F(\beta)$ and the corresponding optimal solutions for different vectors β in a nondecreasing order of the objective function values – which also means a nondecreasing order for the values $F(\beta)$.

As $\mathbf{x}_R = 0$ is not a solution of the original problem, the minimal cost solution may be determined for (4.52)–(4.59) considering all possible right hand sides $\beta \pmod{D}$ in an increasing order of objective function values for the corresponding optimal solutions, until the solution is obtained for

$$\beta = \beta_0.$$

Calculate

$$c'_k = \min_{j \in J_R} c'_j.$$

Then the optimal solution of (4.57)–(4.59) with right hand side β_k is obviously

$$x_k = 1, \ x_j = 0, \qquad (j \neq k).$$

The second, third, etc. cheapest solution may be determined in a similar way, taking into account the immediate descendants of solutions obtained so far, including the **0** vector. The immediate descendant of a vector (solution) is obtained by increasing only one of the components by one unit. The number of candidates may be substantially decreased by using the branch-and-bound principle, the above note (2°), and other ideas discussed in the following section.

The algorithm described here differs from that of Greenberg in the respect that a repetition-free enumeration of optimal solutions for different right hand sides is automatically guaranteed. Some further suggestions may be found after the numerical example and in the next section.

In order to avoid a complicated notation – resulting from the fact that only the subscripts in the set J_R are used in (4.57)–(4.59), the new problem is formulated as

$$\min \sum_{j=1}^{s} d_j y_j \tag{4.62}$$

$$\sum_{j=1}^{s} \gamma_j y_j = \gamma \pmod{D} \tag{4.63}$$

$$0 \leqq y_j \leqq u_j \quad y_j \text{ integer} \qquad (j = 1, 2, \ldots, s) \tag{4.64}$$

where $\mathbf{x}_R = \mathbf{y}$

or using a subscript transformation $t(j)$

$$x_{t(j)} = y_j \qquad (j = 1, 2, \ldots, s).$$

This means, that

$$J_R = \{\mathrm{t}(1), \mathrm{t}(2), \ldots, \mathrm{t}(s)\}$$

and also

$$c'_{t(j)} = d_j \quad \beta_{t(j)} = \gamma_j \quad r_{t(j)} = u_j \qquad (j = 1, \ldots, s) \quad \text{and} \quad \beta_0 = \gamma_0.$$

Problem (4.62)–(4.64) is to be solved for $\gamma = \gamma_0$ is such a way, that

$$\mathbf{r}_B \geqq \mathbf{x}_B = \mathbf{B}^{-1}\mathbf{b} - \mathbf{B}^{-1}\mathbf{R}\mathbf{y} \geqq \mathbf{0} \tag{4.65}$$

where $\mathbf{r}_B = (r_{j_1}, r_{j_2}, \ldots, r_{j_m}) \quad J_B = \{j_1, j_2, \ldots, j_m\}$.

Let us define $h(\mathbf{y})$ as the highest subscript corresponding to a nonzero component of vector $\mathbf{y}$

$$h(\mathbf{0}) = 0$$

$$h(\mathbf{y}) = \{\max j \mid j \in \{1, 2, \ldots, s\}\, y_j > 0\} \quad \text{if} \quad \mathbf{y} \neq \mathbf{0}.$$

Definition: *The set of continuations of a vector* $\tilde{\mathbf{y}}$ *is defined as follows:*

$$\mathcal{C}(\tilde{\mathbf{y}}) = \{\mathbf{y} \mid y_j = \tilde{y}_j \quad \text{if} \quad j < h(\tilde{\mathbf{y}}) \quad \text{and} \quad y_j \geqq \tilde{y}_j \quad \text{if} \quad j \geqq h(\tilde{\mathbf{y}}),\ \mathbf{y} \text{ integer}\}.$$

The vector $\mathbf{y}$ is called a *feasible continuation* of $\tilde{\mathbf{y}}$ relative to problem (4.62)–(4.64) if

$$\mathbf{y} \in \mathcal{C}(\tilde{\mathbf{y}}) \quad \text{and} \quad \mathbf{y} \leqq \mathbf{u}.$$

The set of feasible solutions of (4.62)–(4.64) is equivalent to the feasible continuations of vector **0**.

Any set $\mathcal{C}(\tilde{\mathbf{y}})$ may be decomposed to pairwise disjoint subsets

$$\mathcal{C}(\tilde{\mathbf{y}}) = \mathcal{C}(\mathbf{y}^1) \cup \ldots \cup \mathcal{C}(\mathbf{y}^p) \cup \{\tilde{\mathbf{y}}\} \tag{4.66}$$

where

$$y_j^k = \begin{cases} \tilde{y}_j & \text{if} \quad j \neq j_k = h(\tilde{\mathbf{y}}) + k - 1 \\ \tilde{y}_{j_k} + 1 & \text{if} \quad j = j_k \end{cases} \tag{4.67}$$

$$(k = 1, 2, \ldots, p = s - h(\tilde{\mathbf{y}}) + 1).$$

The vectors $\mathbf{y}^1, \ldots, \mathbf{y}^p$ are called immediate continuations of vector $\tilde{\mathbf{y}}$.

An algorithm for solving (4.62)–(4.65), which is equivalent to the original problem (4.49)–(4.51), is described by using the following notation.

Notation

$\mathbf{y}^k$	solution k already constructed
γ^k	$\mathbf{y}^k$ is a solution of (4.62)–(4.64) with right hand side $\gamma = \gamma^k$
f_k	the cost of solution $\mathbf{y}^k$
$\hat{f}$	the cost of best solution found so far
t	superscript of the next solution to be constructed
P	the set of superscripts corresponding to the active (not yet decomposed) solutions
k^*	the superscript of the solution to be decomposed next
a(k)	denotes the ancestor of solution k, that is, the serial number of solution from which it was generated.

The algorithm itself is described by the following steps.

Step 1: Initialization.

$$P = \{0\} \quad \mathbf{y}^0 = 0, \quad \gamma^0 = 0, \quad f_0 = 0, \quad t = 1, \quad k^* = 0, \quad \hat{f} = +\infty,$$

go to Step 4.

Step 2: If set P is empty, go to Step 7. Otherwise calculate

$$\min_{k \in P} f_k = f_{k^*}.$$

In the case of alternative minima, chose one of them, e.g. the one with the minimal subscript.

Step 3: If $\gamma^{k^*} = \gamma_0$ and $\mathbf{r}_B \geqq \mathbf{x}_B \geqq \mathbf{0}$ which is calculated from equation (4.65) then go to Step 8. Otherwise go to the next step.

Step 4: Delete k^* from set P.

Step 5: Calculate all immediate continuations of vector $\mathbf{y}^k$ by the defining equation (4.67) applied for $\hat{\mathbf{y}} = \mathbf{y}^{k*}$. If none of the continuations is feasible (each of them violates at least one of the upper bounds (4.64) or the value of the solution is not smaller than $\hat{f}$) then go to Step 2. Otherwise continue at the next step.

Step 6A: Denote the feasible immediate continuations of vector $\mathbf{y}^{k*}$ by $\mathbf{y}^k$ ($k = t+1, \ldots, t+n_{k*}$). Add the new superscripts, $t+1, \ldots, t+n_{k*}$ to set P. Calculate the corresponding right hand sides and prices

$$\gamma^k = \gamma^{k*} + \gamma_{l(k)} \quad f_k = f_{k*} + d_{l(k)}$$

where $k = t+1, \ldots, t+n_{k*}$ and $l(k)$ is the subscript of the variable that was increased by 1 when solution $\mathbf{y}^k$ was formed from solution $\mathbf{y}^{k*}$. Increase the value of t by n_{k*}.

Step 6B: If there is a $k \in \{t+1, \ldots, t+n_{k*}\}$ such that

$$\gamma^k = \gamma_0, \quad f_k < \hat{f} \text{ and } \mathbf{0} \leqq \mathbf{x}_B(\mathbf{y}^k) \leqq \mathbf{r}_B .$$

then let $\hat{f} = f_k$. If $\hat{f}$ is decreased, then

$$P := P - \{k \mid f_k \geqq \hat{f}\} .$$

Go to Step 2.

Step 7: No solution of (4.62)–(4.65) or of equivalent problem (4.45)–(4.48) exists. Stop.

Step 8: $\mathbf{x}_B$ and $\mathbf{x}_R = \mathbf{y}^{k*}$ obtained in Step 3 form an optimal solution of (4.45)–(4.48). Stop.

The above algorithm is obviously finite because each variable $y_1, \ldots, y_s$ is bounded from above, thus the number of solutions $\mathbf{y}^k$ generated by the algorithm is also finite. On the other hand, one of the solutions is always deleted at each iteration (going through Steps 2–6), therefore the set P becomes empty in a finite number of steps if the algorithm does not stop at Step 8 beforehand.

The algorithm generates all solutions of the problem in an order of nondecreasing costs thus the first feasible solution with right hand side $\gamma = \gamma_0$ will also be optimal.

Example: Let us consider the following problem of originally inequality form, with the surplus variables already added

$$\begin{aligned} \min\ & 7x_1 + 2x_2 + 3x_3 + 4x_4 + 7x_5 + x_6 + 2x_7 \\ & 2x_1 + 6x_2 + 6x_3 - x_4 + 7x_5 - 5x_6 + 3x_7 - x_8 = 2 \\ & 2x_1 + 3x_2 + x_3 + 5x_5 + x_6 + 3x_7 - x_9 = 3 \end{aligned}$$

$$x_1 - x_2 + x_4 + x_5 + x_6 - x_{10} = 1$$

$$0 \leqq x_j \leqq 2, \text{ integer} \qquad (j = 1, \ldots, 7).$$

It is not a further restriction to assume that

$$0 \leqq x_8 \leqq 48, \quad 0 \leqq x_9 \leqq 27, \quad 0 \leqq x_{10} \leqq 7, \quad x_8, x_9, x_{10} \text{ integers.}$$

Now we solve the problem without the integer restrictions by the simplex method. Then the optimal basis consists of

$$\mathbf{x}_B^T = (x_3, x_5, x_6).$$

The corresponding basis, its inverse and the optimum with the corresponding objective function value

$$\mathbf{B} = \begin{pmatrix} 6 & 7 & -5 \\ 1 & 5 & 1 \\ 0 & 1 & 1 \end{pmatrix} \quad \mathbf{B}^{-1} = \frac{1}{12} \begin{pmatrix} 4 & -12 & 32 \\ -1 & 6 & -11 \\ 1 & -6 & 23 \end{pmatrix} \quad \begin{matrix} x_3 = {}^4/_{12} \\ x_5 = {}^7/_{12} \\ x_6 = {}^5/_{12} \end{matrix} \quad z = {}^{66}/_{12}.$$

Now the problem is transformed into the form (4.57)–(4.59)

$$\begin{aligned} \min \; & 2x_1 + 18x_2 + 12x_4 + 6x_7 + 6x_8 + 42x_{10} \\ & 16x_1 - 44x_2 + 28x_4 - 24x_7 - 4x_8 + 12x_9 - 32x_{10} \equiv 4 \\ & -x_1 + 32x_2 - 10x_4 + 15x_7 + x_8 - 6x_9 + 11x_{10} \equiv 5 \pmod{12} \\ & 13x_1 - 35x_2 + 22x_4 - 15x_7 - x_8 + 6x_9 - 23x_{10} \equiv 7 \\ & 0 \leqq x_j \leqq 2 \quad x_j \text{ integer}, \qquad (j = 1, \ldots, 7) \\ & 0 \leqq x_8 \leqq 22, \quad 0 \leqq x_9 \leqq 12, \quad 0 \leqq x_{10} \leqq 3. \end{aligned}$$

The coefficients of the problem may be reduced (mod 12) to fall within the interval [0, 11].

$$\begin{aligned} \min \; & 2x_1 + 18x_2 + 12x_4 + 6x_7 + 6x_8 + 42x_{10} \\ & 4x_1 + 4x_2 + 4x_4 + 4x_8 + 4x_{10} = 4 \\ & 11x_1 + 11x_2 + 2x_4 + 3x_7 + x_8 + 6x_9 + 11x_{10} = 5 \pmod{12} \\ & x_1 + x_2 + 10x_4 + 9x_7 + 11x_8 + 6x_9 + x_{10} = 7. \end{aligned}$$

The above algorithm may be applied to this problem with $\gamma_0 = (4, 5, 7)$. It should be noted that x_j is also bounded by the rank of γ_j minus one. In our example $x_9 \leqq 1$, because the rank of (0, 6, 6) (mod 12) is 2. Similarly $x_8 \leqq 11$.

Both the original subscripts ($x_{t(j)}$) and the new subscripts (y_j) of nonbasic variables are used for convenience. The other notation signs are explained before the description of the algorithm.

Step 1:

$P = \{0\}$, $\mathbf{y}^0 = 0$, $\gamma^0 = 0$, $f_0 = 0$, $t = 1$, $K^* = 0$, $\hat{f} = +\infty$

Steps 4–6:

$x_{t(j)}$	1	2	4	7	8	9	10
y_j	1	2	3	4	5	6	7
$a(k)$	0	0	0	0	0	0	0
k	1	2	3	4	5	6	7
γ^k	4	4	4	0	8	0	4
	11	11	2	3	1	6	11
	1	1	10	9	11	6	1
f_k	2	18	12	6	6	0	42
	*	X	*	*	*	X	*

$P = \{1, 2, 3, 4, 5, 6, 7\}$

Step 2: $\min_{k \in P} f_k = f_6$, $k^* = 6$

Steps 4–6: As $x_9 \leqq 1$, solution 6 may be extended by x_{10} (y_7) only:

$x_{t(j)}$	10
y_j	7
$a(k)$	6
k	8
γ^k	4 5 7
f_k	42
	X

This is feasible solution of the modified problem, and also for the original problem, because

$$\mathbf{x}_B = \frac{1}{12}\begin{pmatrix} 4 & -12 & 32 \\ -1 & 6 & -11 \\ 1 & -6 & 23 \end{pmatrix}\left[\begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix} - \begin{pmatrix} 0 & 0 \\ -1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right] = \begin{pmatrix} 2 \\ 0 \\ 2 \end{pmatrix} \geqq \mathbf{0}$$

A feasible solution has already been obtained with $\hat{x}^T = (0, 0, 2, 0, 0, 2, 0, 0, 1, 1)$ $\hat{z} = 8, \hat{f} = 42$ and in the modified problem we are to consider solutions only with $f_k < \hat{f} = 42$. However this solution has not been reached in Step 3, therefore it may not be optimal.

$$P = \{1, 2, 3, 4, 5\}.$$

The continuations of the above solution ($k = 8$) should not be considered because it is already a feasible solution. Furthermore, new bounds on the variables may be obtained for some of the nonbasic variables in order to remain under $\hat{z} = 8$: $x_1 \leqq 1, x_4 \leqq 1$.

Step 2:

$$\min_{k \in P} f_k = f_1 \quad k^* = 1$$

Steps 4–6:

$x_{t(j)}$	2	4	7	8	9
y_j	2	3	4	5	6
$a(k)$	1	1	1	1	1
k	9	10	11	12	13
γ^k	8	8	4	0	4
	10	1	2	0	5
	2	11	10	0	7
f_k	20	14	8	8	2
	$x_1 = 1$				
	X	*	*	X	X

$$P = \{2, 3, 4, 5, 9, 10, 11, 12, 13\}.$$

The procedure is continued in exactly the same way therefore only the different added sections will be given. Under the columns the sign * means that the corresponding column is already chosen for continuation. The sign X means that the column should not be continued because it is either feasible or it has no better continuation, than the best one found so far.

$x_{i(k)}$	7	8	9	8	9	7	8	9	7	8	9	9	9	9	9	9
y_i	4	5	6	5	6	4	5	6	4	5	6	6	6	6	6	6
$a(k)$	4	4	4	5	5	11	11	11	3	3	4	14	15	17	10	20
k	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
γ^k	0	8	0	4	8	4	0	4	4	0	4	0	8	4	8	0
	6	4	9	2	7	5	3	8	5	3	8	0	10	8	7	9
	6	8	3	10	5	7	9	4	7	9	4	0	2	4	4	3
f_k	12	12	6	12	6	14	14	8	18	18	12	12	12	12	14	14
	$x_7 = 1$			$x_8 = 1$		$x_1 = 1, x_7 = 1$			$x_4 = 1$							
	*	*	X	*	X	X	*	X	X	X	X	X	X	X	X	X

Two further feasible solutions have been found. One of them ($k = 19$) results in an infeasible solution for the original problem

$$\mathbf{x}_B = \frac{1}{12}\begin{pmatrix} 4 & -12 & 32 \\ -1 & 6 & -11 \\ 1 & -6 & 23 \end{pmatrix}\left[\begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix} - \begin{pmatrix} 2 & 3 \\ 2 & 3 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 2 \end{pmatrix}\right] = \begin{pmatrix} 3 \\ -2 \\ 2 \end{pmatrix} \ngeq \mathbf{0}.$$

But the other one ($k = 22$) provides a new feasible solution also for the original problem

$$x_B = \frac{1}{12}\begin{pmatrix} 4 & -12 & 32 \\ -1 & 6 & -11 \\ 1 & -6 & 23 \end{pmatrix}\left[\begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix} - \begin{pmatrix} -1 & 3 \\ 0 & 3 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix}\right] = \mathbf{0}.$$

Now the best solution found so far with the objective function value is

$$\hat{\mathbf{x}}^T = (0, 0, 0, 1, 0, 0, 1, 0, 0, 0) \quad \hat{z} = 6, \quad \hat{f} = 18.$$

Only corrections with $f_k < 18$ are to be calculated. This feasible solution turns out to be optimal after generating a few more solutions for different right hand sides.

Other methods for decreasing the amount of computation will be discussed in the following sections. In the numerical example a new feature was introduced to the algorithm. Not only is the minimal cost solution examined at Step 3, if it provides a feasible solution of the problem, but so are the newly generated solutions. As soon as a feasible solution is found for the original problem (4.45)–(4.48), i.e. besides obtaining γ_0, constraint (4.65) is also satisfied. Then the following correction may be introduced.

The present value of the objective function (4.45), $\hat{z}$ is an upper bound for future feasible solutions. This may provide stronger upper bounds for the nonbasic variables, i.e. for the variables of our modified problem (4.62)–(4.64). On the other hand, it is also possible to take into account the constraint

$$\mathbf{c}^T\mathbf{x} < \hat{z}$$

in general. In the numerical example it was unnecessary to generate the continuations of solution 11 (solutions 19, 20 and 21), for example, because $x_1 = 1$, $x_7 = 1$ corresponds to $z = 9 > \hat{z}$.

This modification may easily be introduced for the algorithm – which was presented in a simple form for easier understanding.

4.7 Further improvement of the single optimum function method

The purpose of the present section is to introduce some modifications to the algorithm of Section 4.6 in order to decrease the amount of computation and the memory requirement. These improvements are partly based on the work of Glover (1969). Glover's results, however, refer only to the nonbounded problem

$$\min \sum_{j=1}^{s} d_j y_j \tag{4.68}$$

$$\sum_{j=1}^{s} \gamma_j y_j = \gamma \tag{4.69}$$

$$y_j \geqq 0, \text{ integer} \qquad (j = 1, \ldots, s), \tag{4.70}$$

where γ, γ_j $(j = 1, \ldots, s)$ are taken from a finite Abelian group G. The generalization toward groups instead of vectors mod D does not mean additional difficulties, therefore our discussion will remain in the space γ, $\gamma_j \in R^m$ (mod D). The existence or nonexistence of explicit upper bounds will, on the other hand, make a great difference therefore we shall follow both directions. It should be noted that the upper bounds restrict the application of some procedures if they are smaller than the natural upper bounds:

$$y_j \leqq r_j - 1 \qquad (j = 1, \ldots, s)$$

where r_j is the rank of the coefficient vector γ_j i.e.

$$r_j\gamma_j = 0 \qquad (\text{mod } D)$$

and no smaller multiplier of γ_j has the same property.

Let us examine first the notion of dominated solutions, which was introduced by Glover for the unbounded problem (4.68)–(4.70). Using the notation of the foregoing section let us consider two solutions $\mathbf{y}^k$ and $\mathbf{y}^l$. Then

$$\gamma^k = \sum_{j=1}^{s} \gamma_j y_j^k, \qquad f_k = \sum_{j=1}^{s} d_j y_j^k$$

$$\gamma^l = \sum_{j=1}^{s} \gamma_j y_j^l, \qquad f_l = \sum_{j=1}^{s} d_j y_j^l.$$

Lemma 4.1: *If $\gamma^k = \gamma^l$ and $f_k < f_l$ for two solutions of the unbounded problem* (4.68)–(4.70) *then the continuations of solution l may be deleted without altering the optimum of the problem.* In this case solution l is said to be dominated by solution k.

Proof: The statement of the lemma is an immediate consequence of the optimality principle. No continuation of y^1 can be optimal for any right hand side. Namely let

$$\mathbf{y}^1 = \mathbf{y}^l + \mathbf{y}^2,$$

then obviously

$$\mathbf{y}^3 = \mathbf{y}^k + \mathbf{y}^2$$

gives the same right hand side $\gamma^1 = \gamma^3$ and $f_1 > f_3$.†

The same statement does not hold for the bounded case without any further restriction. Let us introduce the notation J_u for the set of subscripts, that have an upper bound u_j smaller than the above natural upper bound:

$$J_u = \{j \mid u_j < r_j - 1, \quad j = 1, \ldots, s\}.$$

Then the following assertion may be used.

Lemma 4.2: *If $\gamma^k = \gamma^l$, $f_k < f_l$ and $y_j^k \leqq y_j^l$, $j \in J_u$ for the solutions k* and *l of the bounded problem* ((4.62)–(4.64) *then solution l is dominated by solution k, that is, continuations of solution l may be deleted.*

Proof: As an addition to the justification of the previous lemma, we have to see only that if $\mathbf{y}^1$ satisfies the upper bound restriction, then so does $\mathbf{y}^3$ defined above or another vector $\tilde{\mathbf{y}}^3$ can be constructed with this property, which also dominates solution $\mathbf{y}^1$.

For this purpose let us define

$$\tilde{y}_j^3 = y_j^3 - k_j r_j$$

where r_j is the rank of γ_j and $k_j \geqq 0$ integer is defined so that

$$0 \leqq \tilde{y}_j^3 \leqq r_j - 1.$$

† It should be noted that $\mathbf{y}^3$ is not necessarily a continuation of vector $\mathbf{y}^k$, but this fact does not make the proof invalid.

Now

$$\hat{y}_j^3 \leqq r_j - 1 \leqq u_j \qquad \text{for} \qquad j \notin J_u$$

$$\hat{y}_j^3 \leqq y_j^k + y_j^2 \leqq y_j^l + y_j^2 = y_j^1 \leqq u_j \qquad \text{for} \qquad j \in J_u$$

and

$$\hat{y}^3 = \gamma^3 = \gamma^1 \qquad \tilde{f}_3 \leqq f_3 < f_1,$$

that is, if $\mathbf{y}^1$ provides a feasible solution for γ^1, then so does $\hat{\mathbf{y}}^3$ with a better objective function value.

Note 1°: In the proofs of the foregoing lemmata it was not utilized that $\mathbf{y}^1$ is a continuation of $\mathbf{y}^l$ in the sense defined in Section 4.6, only the fact that $\mathbf{y}^2 \geqq 0$ integer. In other words, both lemmata may be extended as a means of disregarding all solutions with

$$\mathbf{y} \geqq \mathbf{y}^l$$

if the conditions are met. The checking of this inequality, however, is not very easy throughout the algorithm if the number of dominated solutions is sufficiently large. There are two exceptions. One of them is the above-mentioned set of continuations which will be disregarded automatically by closing the solution $\mathbf{y}^l$. The other case is that in which only one component of $\mathbf{y}^l$ is nonzero. Then we have obtained a new upper bound for that particular component.

Note 2°: In most cases it is substantially more difficult to solve the upper bounded problem relative to the one without explicit upper bounds. From Lemma 4.2, it is also clear that the smaller the number of upper bounded variables the easier it is to solve. For this reason it may be practical to solve the unbounded problem even in the case of bounded variables, and to introduce the upper bounds for only those variables exceeding their permitted maximal value. If some additional variables exceed the upper bound, further iteration is necessary, but each time the previous calculations may be utilized.

Another way of accelerating the algorithm is also described by Glover and may be applied both to the unbounded and to the bounded problem. The basic idea may be outlined as follows. Instead of stopping the algorithm when the optimal solution $\mathbf{y}$ has already been generated, it is sufficient to obtain solutions $\mathbf{y}^k$ and $\mathbf{y}^l$, if

$$\hat{\mathbf{y}} = \mathbf{y}^k + \mathbf{y}^l \qquad \text{and thus} \qquad \gamma_0 = \gamma^k + \gamma^l. \tag{4.71}$$

As the solutions are generated in a nondecreasing order of the objective function value the pair $\mathbf{y}^k$, $\mathbf{y}^l$ satisfying (4.71) will be generated first, for which

$$|\mathbf{d}^T\mathbf{y}^k - \mathbf{d}^T\mathbf{y}^l| = \min_{\mathbf{y}' + \mathbf{y}'' = \hat{\mathbf{y}}} |\mathbf{d}^T\mathbf{y}' - \mathbf{d}^T\mathbf{y}''|. \tag{4.72}$$

In other words, the procedure is simply to check after each new $\mathbf{y}^k$ and γ^k generated, whether the solution ($\mathbf{y}^l$) for $\gamma_0 - \gamma^k$ is already available or not. In the case of a positive answer, $\mathbf{y}^k + \mathbf{y}^l$ provides a new feasible solution.

It remains for us to establish a stopping rule. For this purpose let us denote by N_r the next value to be chosen in the dynamic recursion, in the notation of the foregoing section

$$N_r = \min_{k \in P} f_k .$$

Furthermore let

$$d^* = \max d_j$$

$$j = 1, \ldots, s .$$

Lemma 4.3: *The optimum of the bounded problem* (4.62)–(4.64) *or the unbounded problem* (4.68)–(4.70) *has been obtained if a pair of solutions* $\mathbf{y}^k$, $\mathbf{y}^l$ *is found in such a way that* (*supposing* $f_k \geqq f_l$)

$$\gamma^k + \gamma^l = \gamma_0$$

and

$$N_r - f_l > d^* .$$

Proof: Let us suppose, on the contrary, that no optimal solution has been generated yet.

Let us consider any pair of solutions $\mathbf{y}^1$, $\mathbf{y}^2$, for which

$$\mathbf{y}^1 + \mathbf{y}^2 = \tilde{\mathbf{y}}$$

and the least one of the solutions (say $\mathbf{y}^2$), is not yet generated, $\tilde{\mathbf{y}}$ is an optimal solution and the pair $\mathbf{y}^1$, $\mathbf{y}^2$ satisfies minimality requirement (4.72). Then

$$d^* \geqq f'' - f' \geqq N_r - f_l > d^*,$$

which is a contradiction. The lemma is thus proved.

When using the algorithm for solving a problem of type (4.45)–(4.48) the above lemma naturally does not provide a stopping rule when the optimal solution is obtained, until the optimal solution of the modified problem (4.62)–(4.64) is available – which may not be a feasible solution for the original problem.

4.8. Application of the shortest path algorithm in dynamic optimization

In this section Shapiro's algorithm (1968a) will be discussed and completed by a method for group decomposition. Now let us restate the unbounded problem for convenience

$$\min \sum_{j=1}^{s} d_j y_j \tag{4.73}$$

$$\sum_{j=1}^{s} \gamma_j y_j = \gamma \pmod{D} \tag{4.74}$$

$$y_j \geqq 0 \text{ integer} \qquad (j = 1, \ldots, s) . \tag{4.75}$$

This problem is to be solved for $\gamma = \gamma_0$ (it is the same as (4.62)–(4.64), but without upper bounds). The unbounded version of the original problem (4.45)–(4.48) is also solved if the nonnegativity of the basic variables holds:

$$\mathbf{x}_B = \mathbf{B}^{-1}\mathbf{b} - \mathbf{B}^{-1}\mathbf{R}\mathbf{y} \geqq \mathbf{0}. \tag{4.76}$$

In the present section only (4.73)–(4.75) will be considered. First of all we attempt to reduce the number of rows by determining a basis for the group G generated by the column vectors $\gamma_1, \gamma_2, \ldots, \gamma_s$. It should be noted that G is nothing but the factor group of the following two groups

$$M = \left\{ \sum_{j=1}^{s} \gamma_j y_j : y_j \text{ integer} \right\}$$

$$N = \left\{ \sum_{j=1}^{m} De_j y_j : y_j \text{ integer} \right\} \cap M,$$

that is,

$$G = M/N.$$

As is well known from abstract algebra, every finite Abelian group can be decomposed to a direct sum of cyclic groups. Or another possible approach is the well known theorem of elementary divisors first proved by Smith (1861) which states, in our case, that the above group G is isomorphic with the direct sum of the following groups

$$G \cong Z_{q_1} \oplus Z_{q_2} \ldots \oplus Z_{q_r}$$

where Z_{q_i} is the residue class of integers modulo q_i and $q_1, \ldots, q_r$ are integers such that $q_1 \mid q_2 \mid \ldots \mid q_r$ (q_i divides q_{i+1}) and $D = q_1 q_2, \ldots, q_r$.

A simple consequence of these assertions is that group G has D elements.

The other important consequence is that instead of an m-constraint problem it is sufficient to deal with an r-constraint problem. This r may often be 1.

Now, instead of the above decomposition another one will be suggested which is easy to construct and it seems to have some advantages from the practical point of view.

Let us consider (4.73)–(4.75) and the above group G, generated by the coefficient vectors $\gamma_1, \ldots, \gamma_s$. Let us denote a subset of these vectors which form a basis in G, by $\gamma_{k_1}, \ldots, \gamma_{k_p}$ and the product of ranks

$$r(\gamma_{k_1}), \ldots, r(\gamma_{k_p}) = D.$$

It will be constructively shown later, that such a subset exists. By the definition of the basis there exist nonnegative integer coefficients α_{ij} $(i = 1, \ldots, p; j = 1, \ldots, s)$ such th at

$$\gamma_j \equiv \sum_{i=1}^{p} \alpha_{ij} \gamma_{k_i} \pmod{D} \qquad (j = 1, \ldots, s). \tag{4.77}$$

Substituting the expression (4.77) into the congruence (4.74) and denoting the coefficient vector $\alpha_j^T = (\alpha_{1j}, \alpha_{2j}, \ldots, \alpha_{pj})$ we obtain the following new form of the problem:

$$\min \sum_{j=1}^{s} d_j y_j \tag{4.78}$$

$$\sum_{j=1}^{s} \alpha_j y_j \equiv \alpha \pmod{\mathbf{D}} \tag{4.79}$$

$$y_j \geqq 0, \quad \text{integer} \qquad (j = 1, \ldots, s) \tag{4.80}$$

which has exactly the same form as (4.73)–(4.75). This problem should be solved for $\alpha = \alpha_0$ where α_0 is the coordinate vector of γ_0 in the basis $\alpha_{k_1}, \ldots, \alpha_{k_p}$. In the group problem (4.78)–(4.80), $\mathbf{D} = (r_1, \ldots, r_p)$ is used as modulus vector; in other words in congruence i of system (4.79) the elements are understood mod $r_i = r(\gamma_{k_i})$.

The advantage of (4.78)–(4.80) compared with (4.73)–(4.75) is that it has usually substantially less – and never more – rows. It often happens that (4.78)–(4.80) has only a single row. In dynamic optimization the number of constraint rows determines the difficulty of the problem. Another great advantage is the much simpler bookkeeping and the less memory necessary, resulting from the simpler representation. The methods described in the previous sections can be applied to (4.78)–(4.80) as well.

Now an entirely different approach – introduced by Shapiro (1968a) – will be discussed.

If only the unbounded problem (4.78)–(4.80) is to be solved, then we can suppose that there are no identical column vectors among $\alpha_1, \ldots, \alpha_s$ because in the opposite case only one of the identical vectors (the one with minimal cost coefficient) should be kept. Similarly there is no need to keep variables y_j with 0 coefficient vector.

The basic idea of solving (4.78)–(4.80) is the use of an algorithm similar to the one for solving shortest route problems. For this purpose the shortest route is determined for each $\alpha \in G$ in lexicographic order using an iterative procedure. In this algorithm $H_k(\alpha)$ will mean the length of the shortest path from Θ (zero element of the group) to α at iteration k and $j_k^*(\alpha)$ is the subscript of last edge in this shortest route (before α).

The functions $H_q(\alpha)$ $(q = 0, 1, \ldots)$ are defined recursively as follows:

$$H_q(\Theta) = 0 \qquad (q = 0, 1, 2, \ldots)$$

$$H_0(\alpha) = \infty \qquad (\alpha \in G, \alpha \neq 0) \tag{4.81}$$

$$H_q(\alpha) = \min \begin{cases} \min \{d_j + H_q \quad (\alpha - \alpha_j) \mid j = 1, \ldots, s; \quad \alpha - \alpha_j \prec \alpha\} \\ \min \{d_j + H_{q-1}(\alpha - \alpha_j) \mid j = 1, \ldots, s; \quad \alpha \prec \alpha - \alpha_j\} \end{cases}$$

where the meaning of the sign $\prec$ is "proceeds" or "lexicographically smaller than". Obviously the substraction $\alpha - \alpha_j$ is understood component-wise, mod r_i; where r_i is the rank of the ith element of the basis, because the components of the α vectors

are in the space spanned by this basis. For this reason the above calculation of H_q functions should be performed for all α

$$0 \leqq \alpha_i \leqq r_i - 1, \quad (i = 1, 2, \ldots, p).$$

The function $H_q(\alpha)$ slightly differs from the shortest path from 0 to α using at most q arcs. The difference is in the upper half of the defining equation, i.e. the elements $\alpha - \alpha_j$ for which the iteration q was already performed, are taken into account with their *new* (iteration q) shortest path.

It is easy to see that the above functions $H_q(\alpha)$ should be calculated until

$$H_q(\alpha) = H_{q-1}(\alpha) \qquad \text{for all} \qquad \alpha \in G$$

and furthermore, that this will happen for a $q \leqq D$. The limit function, the shortest path, is denoted by $H(\alpha)$ and the corresponding last node index function by $j^*(\alpha)$.

It is obvious from the principle of optimality that the shortest route function provides the optimum of problem (4.78)–(4.80) for each $\alpha \in G$ thus also for the desired value $\alpha = \alpha_0$ and the corresponding optimal solutions may easily be determined by calculating the path

$$j_1 = j^*(\alpha), j_2 = j^*(\alpha_{j_1}), \ldots, j_k = j^*(\alpha_{j_{k-1}}) \quad \text{until } j_k = 0.$$

The corresponding solution in terms of y_j may be obtained from $\mathbf{y} = \mathbf{0}$ through increasing component $j_1, j_2, \ldots$, etc. by 1. (Some components may be increased several times.)

Example: Solve the following group problem

$$\min 2y_1 + 5y_2 + y_3 + 2y_4 + 4y_5 + 2y_6$$

$$\begin{pmatrix}1\\0\end{pmatrix} y_1 + \begin{pmatrix}0\\1\end{pmatrix} y_2 + \begin{pmatrix}1\\3\end{pmatrix} y_3 + \begin{pmatrix}1\\5\end{pmatrix} y_4 + \begin{pmatrix}0\\2\end{pmatrix} y_5 + \begin{pmatrix}1\\2\end{pmatrix} y_6 \equiv \begin{pmatrix}1\\4\end{pmatrix} \mod \begin{pmatrix}2\\6\end{pmatrix}.$$

As the rank of the elements are 2, 6, 2, 6, 3, 6 an upper bound of the objective function is

$$1.2 + 5.5 + 1.1 + 5.2 + 2.4 + 5.2 = 56.$$

Then $H_q(\Theta) = 0$ $(q = 1, 2, \ldots)$ where $\Theta = \begin{pmatrix}0\\0\end{pmatrix}$ is the zero element of the group.

$$H_0(\alpha) = 57 \qquad \text{for} \qquad \alpha \in G, \quad \alpha \neq \Theta.$$

Let us denote the coefficient vector of y_j by α_j. Then the function $H_1(\alpha)$ is calculated by the expression (4.81) and $j_1^*(\alpha)$ is the subscript of α_j for which the minimum is obtained in the defining equation (4.81). This is repeated for the functions $H_2(\alpha)$ and $j_2^*(\alpha)$, etc.

k	Node	$j_1^*(\alpha)$	$H_1(\alpha)$	$j_2^*(\alpha)$	$H_2(\alpha)$
1	(0, 0)	0	0	0	0
2	(0, 1)	2	5	4	4
3	(0, 2)	5	4	3	3
4	(0, 3)	2	9	1	3
5	(0, 4)	5	8	4	4
6	(0, 5)	2	13	3	3
7	(1, 0)	1	2	1	2
8	(1, 1)	4	6	3	5
9	(1, 2)	6	2	6	2
10	(1, 3)	3	1	3	1
11	(1, 4)	2	6	3	5
12	(1, 5)	4	2	4	2

As $H_3(\alpha) = H_2(\alpha)$ for all $\alpha \in G$, there is no need to continue the calculation: $H(\alpha) = H_2(\alpha)$, $j^*(\alpha) = j_2^*(\alpha)$. The optimal solution for

$$\alpha = \begin{pmatrix} 1 \\ 4 \end{pmatrix}$$

can easily be calculated

$$j_1 = j^*(\alpha) = 3, \quad j_2 = j^*((0, 1)) = 4, \quad j_2 = j^*((1, 2)) = 6$$

thus the optimal solution is

$$y_3 = y_4 = y_6 = 1 \qquad \text{with} \qquad \hat{z} = 5$$

Now let us return to the question of determining a basis in group G among the given vectors $\gamma_1, \ldots, \gamma_s$. Denote the rank of the element γ in group G by $r(\gamma)$: (see Note 1°).

Let us consider the following procedure.

Step 1: Denote

$$\gamma_1^0 = \gamma_1, \quad \gamma_2^0 = \gamma_2, \ldots, \gamma_s^0 = \gamma_s, \quad k = 0.$$

Step 2: Chose any subscript i (see Note 4°) with the property that at least two of the elements

$$\gamma_{i1}^k, \gamma_{i2}^k, \ldots, \gamma_{is}^k$$

are nonzero (mod D). If no such row i exists go to Step 6, otherwise to the next step.

Step 3: If any of these elements divides the others (mod D), then denote the column (second) subscript of this element by j_k and go to Step 5. In the opposite case go to the next step.

Step 4: Determine the greatest common divisor of the elements appearing in the previous step in the following form (see Note 5°)

$$d \equiv \lambda_1^{k+1}\gamma_{i_1}^k + \lambda_2^{k+1}\gamma_{i_2}^k + \ldots + \lambda_s^{k+1}\gamma_{i_s}^k \pmod{D}$$

and define a new column

$$\gamma_{s+k}^k = \sum_{j=1}^{s} \lambda_j^{s+1}\gamma_j^k .$$

Let $j_k = s + 1$ increase the value of s by 1 and go to the next step.

Step 5: Compute for all $j \neq j_k$

$$\gamma_j^{k+1} \equiv \gamma_j^k - \lambda_j^{k+1}\gamma_{j_k} \pmod{D}, \quad \text{where } \gamma_{ij} \equiv \gamma_{ijk} \pmod{D}$$

and

$$\gamma_{j_k}^{k+1} = \gamma_{j_k}^k$$

go to Step 2.

Step 6: The procedure is finished. The vectors γ_j^k with

$$r(\gamma_j^k) > 1$$

provides a basis and

$$\prod_{j=1}^{s} r(\gamma_j^k) = D .$$

The finiteness of the algorithm is obvious, because at each iteration (Step 3–Step 5) the number of rows containing only a single nonzero element is increased by at least one. Thus the condition of Step 2 is satisfied after a finite number of steps. On the other hand, after performing an iteration, the new column vector still generates the entire group G because only the multiples of one vector are substracted from the others. Thus a basis is constructed at the end of the procedure.

All coefficient vectors γ_j may easily be expressed by this new basis because the basis vectors have no two nonzero elements in a row.

Note 1°: The rank of a vector $\gamma_j \in G$ may easily be determined as the least common multiple of integers λ_i $(i = 1, \ldots, m)$, where

$$\lambda_i\gamma_{ij} \equiv 0 \pmod{D} \quad 0 < \lambda_i < D \quad (i = 1, \ldots, m)$$

and no smaller integer has the same property.

Note 2°: It is sufficient to start with the columns of $\mathbf{D} \cdot \mathbf{B}^{-1}$, instead of using all γ_j, because coefficient vectors γ_j are integer combinations of these columns.

Note 3°: If we want to keep the original vectors γ_j in the group basis, then the following procedure may be suggested: Determine the element γ_{j_1} with the highest rank r_{j_1}. Delete all columns which are multiples of column γ_{j_1}.

Choose the vector with highest rank in the remaining columns: γ_{j_3}. Delete all columns that may be expressed as integer linear combinations of γ_{j_1} and γ_{j_2}, etc. Continue until all elements are either in the basis or deleted. If

$$\prod_{k=1}^{P} r_{jk} = D. \tag{4.82}$$

then the decomposition is finished. In the opposite case there exist $\lambda_1, \ldots, \lambda_p$ such that

$$\sum_{k=1}^{P} \lambda_k \gamma_{jk} \equiv 0 \,(\mathrm{mod}\, D) \quad 0 \leqq \lambda_k < r_k \quad (k = 1, \ldots, p) \tag{4.83}$$

and

$$\sum_{k=1}^{P} \lambda_k > 0.$$

Then the above procedure may be applied for all γ_j $(j = 1, \ldots, m)$ or sufficiently only for those with $\lambda_k > 0$ in expression (4.83). In the latter case the method should be continued until the equation (4.82) is satisfied.

Note 4°: It is practical to chose row i with the smallest value of greatest common divisors.

Note 5°: The greatest common divisor of integers $n_1, n_2, \ldots, n_s$ may be determined by the usual way of dividing each number by the smallest one of them n_{k_1}. Then each number (except n_{k_1} itself) is substituted by the corresponding residua. The procedure is continued until the smallest nonzero element in the last row divides the others which is then the greatest common divisor. The method may also be used to express the greatest common divisor as the integer combination of the elements $n_1, \ldots, n_s$. At the end the multipliers are taken mod D.

Example: Calculate the greatest common divisor and the corresponding expression (mod 1000) for the numbers 210, 330 and 462.

Residua			Quotients (x denotes the smallest)		
c_1	c_2	c_3			
210	330	462	x	1	2
210	120	42	5	2	x
0	36	42		x	1
0	36	6			

The multiplications and substractions may easily be followed by matrix notation:

$$(c_1, c_2, c_3)\begin{pmatrix} 1 & -1 & -2 \\ & 1 & \\ & & 1 \end{pmatrix}\begin{pmatrix} 1 & & \\ & & 1 \\ -5 & -2 & 1 \end{pmatrix}\begin{pmatrix} 1 & & \\ & 1 & -1 \\ & & 1 \end{pmatrix} = (c_1, c_2, c_3)\begin{pmatrix} 11 & 3 & -5 \\ 0 & 1 & -1 \\ -5 & -2 & 3 \end{pmatrix} =$$

$$= (11c_1 - 5c_3, \quad 3c_1 + c_2 - 2c_3, \quad -5c_1 - c_2 + 3c_3).$$

As the greatest common divisor has appeared in the third column

$$d = -5c_1 - c_2 + 3c_3$$

and indeed

$$6 = -5\times 210 - 330 + 3\times 462$$

$$6 \equiv 995\times 210 + 999\times 330 + 3\times 462 \pmod{1000}.$$

Now let us see a numerical example for determining the decomposition of a group.

Example: Determine a basis in the group generated by the following vectors, together with the coordinates of all vectors in the new system (mod 36)

$$\begin{matrix} \gamma_1 & \gamma_2 & \gamma_3 & \gamma_4 & \gamma_5 & \gamma_6 & \gamma_7 \\ \begin{bmatrix}27\\24\\18\\0\\9\end{bmatrix} & \begin{bmatrix}18\\24\\12\\24\\6\end{bmatrix} & \begin{bmatrix}18\\0\\0\\0\\18\end{bmatrix} & \begin{bmatrix}27\\12\\30\\24\\33\end{bmatrix} & \begin{bmatrix}0\\12\\24\\12\\12\end{bmatrix} & \begin{bmatrix}18\\0\\24\\12\\30\end{bmatrix} & \begin{bmatrix}0\\0\\12\\24\\24\end{bmatrix} \end{matrix}$$

According to note 4°, we calculate the greatest common divisor (GCD) in each row: 9, 12, 6, 12, 3. Thus the GCD of the 5th row is generated by introducing the new vector

$$\gamma_8 = \gamma_1 - \gamma_2.$$

By substracting multiples of the vector

$$\gamma_8^T = (9, 0, 6, 12, 3) \qquad r(\gamma_8) = 12$$

we obtain zero vectors except the second component. Then the vector

$$\gamma_4^{1T} = (0, 12, 0, 0, 0) \qquad r(\gamma_4^1) = 3$$

may be used to eliminate the remaining nonzero numbers, as 12 is the GCD in this row. Now the new "coordinates" of our vectors are

$$\begin{matrix} & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 & \alpha_6 & \alpha_7 \\ \gamma_8 & \begin{pmatrix}3\\2\end{pmatrix} & \begin{pmatrix}2\\2\end{pmatrix} & \begin{pmatrix}6\\0\end{pmatrix} & \begin{pmatrix}11\\1\end{pmatrix} & \begin{pmatrix}4\\1\end{pmatrix} & \begin{pmatrix}10\\0\end{pmatrix} & \begin{pmatrix}8\\0\end{pmatrix} \\ \gamma_4^1 & & & & & & & \end{matrix} \qquad \left(\bmod \begin{pmatrix}12\\3\end{pmatrix} \right)$$

which means that the components are understood mod 12 and mod 3 respectively.

4.9 Extension of group theoretic algorithms to general linear integer programming problems

In the first part of the section we shall consider the unbounded problem

$$\min \mathbf{c}^T\mathbf{x} \tag{4.84}$$

$$\mathbf{Ax} = \mathbf{b} \tag{4.85}$$

$$x_j \geqq 0, \quad \text{integer} \qquad (j = 1, \ldots, n). \tag{4.86}$$

The discussion is partly based on the work of Shapiro (1968b). The main idea is to solve the corresponding group problem

$$\min \sum_{j=1}^{n} d_j y_j \tag{4.87}$$

$$\sum_{j=1}^{s} \alpha_j y_j \equiv \alpha_0 \,(\text{mod } \mathbf{D}) \tag{4.88}$$

$$y_j \geqq 0, \text{ integer} \qquad (j = 1, \ldots, s) \tag{4.89}$$

as is described in Section 4.8 and check the nonnegativity constraints for the basic variable:

$$\mathbf{x}_B = \mathbf{B}^{-1}\mathbf{b} - \mathbf{B}^{-1}\mathbf{R}\mathbf{y} \geqq \mathbf{0}. \tag{4.90}$$

If the inequalities (4.90) hold, then (4.84)–(4.86) has also been solved. In the opposite case a search routine is applied. Any implicit enumeration method can be the basis for such a search routine. Shapiro (1968b) applies the method described in Lemke and Spielberg (1967), which is rather a branch-and-bound routine. It may be characterized by the following:

(a) The corrections $\mathbf{y}$ are examined in the order corresponding to the sum

$$\sum_{j=1}^{a} y_j = K$$

(K-list). In other words, the $(K + 1)$-list is examined only if the K-list is already empty.

(b) Each correction $\mathbf{y}$ is attempted to be fathomed first, i.e. an effort is made to show that neither $\mathbf{y}$ nor its continuations may result in a feasible solution for (4.87)–(4.90) which is better that the best solution found so far.

(c) If a correction $\mathbf{y}$ cannot be fathomed, then it is deleted from its K-list and the corrections

$$\mathbf{y} + \mathbf{e}_j \qquad (j = 1, 2, \ldots, s)$$

are added to the $(K + 1)$-list.

(d) The search procedure is finished if all K-lists are empty.

Several tests are used for fathoming. Let us denote the best solution of (4.87)–(4.90) found so far by $\mathbf{y}^1$ with the corresponding objective function value $z(\alpha_0)$. Furthermore $z(\alpha)$ denotes the optimum of (4.87)–(4.89) with right hand side α in congruence (4.89) and where $\mathbf{y}^*(\alpha)$ stands for the corresponding optimal solution.

The correction $\mathbf{y}$ is obviously fathomed if at least one of the following conditions is satisfied

(i) If $\mathbf{y}$ is a feasible solution of (4.87)–(4.90), because no continuation of $\mathbf{y}$ can have a better objective function value ($\mathbf{d} \geqq \mathbf{0}$).

(ii) If

$$\mathbf{d}^T\mathbf{y} + z\left(\alpha_0 - \sum_{j=1}^{s} \alpha_j y_j\right) \geqq \hat{z}(\alpha_0).$$

(iii) If

$$\mathbf{y} + \mathbf{y}^*\left(\alpha_0 - \sum_{j=1}^{s} \alpha_j y_j\right)$$

satisfies the nonnegativity requirements (4.90). In this case the optimal continuation results in a feasible solution. No other continuation can have a better objective function value.

An additional feature may be used in these calculations. The algorithm described in Section 4.8 works for any dual feasible basis of (4.84)–(4.86). The optimal basis is used only because in this case $\mathbf{B}^{-1}\mathbf{b} \geqq \mathbf{0}$, and this fact makes the constraints (4.90) easier to satisfy. For the same reason, after increasing the lower bounds for the components of the vector $\mathbf{y}$ it may be useful to solve the continuous version of (4.84)–(4.86) with the corresponding lower bounds added. The new group problem should again be solved. Then the above fathoming tests may be applied for several bases at the same time.

Now let us examine the bounded problem – which may be transformed to a 0–1 problem

$$\min \mathbf{c}^T\mathbf{x} \tag{4.91}$$

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{4.92}$$

$$0 \leqq x_j \leqq 1 \qquad (j = 1, \ldots, n) \tag{4.93}$$

$$x_j \quad \text{integer} \qquad (j = 1, \ldots, n). \tag{4.94}$$

The corresponding group problem may be the unbounded (4.87)–(4.89) or the bounded problem

$$\min \sum_{j=1}^{s} d_j y_j \tag{4.95}$$

$$\sum_{j=1}^{s} \alpha_j y_j = \alpha_0 \,(\mathrm{mod}\, \mathbf{D}) \tag{4.96}$$

$$y_j \in \{0, 1\} \qquad (j = 1, \ldots, s) \tag{4.97}$$

In both cases the restriction for the basic variables

$$1 \geqq \mathbf{x}_B = \mathbf{B}^{-1}\mathbf{b} - \mathbf{B}^{-1}\mathbf{R}\mathbf{y} \geqq \mathbf{0} \tag{4.98}$$

should be used.

A branch-and-bound method can be applied to solve this problem. The basic idea is to build up the usual solution tree by fixing alternatively any of the variables y_j at the value 0 and 1 respectively. For the optimum of both problems an estimation may be obtained either by solving the unbounded problem (4.87)–(4.89) using the algorithm of Section 4.8 or by solving the bounded problem (4.95)–(4.97) using the algorithm of Section 4.6. After obtaining a new separation, one of the terminal branches is chosen for further splitting, e.g. the one with the minimal estimated cost.

The choice of variable for splitting is also important. A simple rule may be to have the maximal difference between the estimations of the branches.

Neither of the above two procedures seems to be computationally feasible in the simple form described here, as opposed to the two algorithms of the foregoing two sections, which are already quite well elaborated. Different, more complex and powerful versions of the above algorithms are under investigation.

CHAPTER 5

A MULTIPHASE DUAL ALGORITHM

5.1 Introduction

The present chapter is based on the work of Glover (1965). This is an enumeration algorithm and it is discussed in more detail than that in Chapter 2. Several further tests of Zontendijk (1970) and others are also introduced, including the strongest surrogate constraints of Geoffrion (1969).

The chapter is completed by a numerical example solved in detail by the multiphase dual algorithm.

5.2 Surrogate constraints

Let us consider *Problem (i)*:

$$\min \mathbf{c}^T\mathbf{x} \tag{5.1}$$

$$\mathbf{A}\mathbf{x} \geqq \mathbf{b} \tag{5.2}$$

$$x_j \in \{0, 1\} \qquad (j = 1, \ldots, n) \tag{5.3}$$

where matrix $\mathbf{A}$ is of size $m \times n$, the vectors $\mathbf{c} \geqq \mathbf{0}$; and $\mathbf{b}$, $\mathbf{x}$ have the corresponding number of components n and m respectively. The nonnegativity of vector $\mathbf{c}$ does not restrict the generality (as was seen in Chapter 1). For simplicity, all the coefficients are supposed as being integers. Among the constraints, the so-called objective function constraint is also included and always updated

$$-\mathbf{c}^T\mathbf{x} \geqq -z_0 + 1$$

where z_0 is the objective function of the best solution obtained so far. At the beginning, if no feasible solution is known, then

$$z_0 = 1 + \sum_{j=1}^{n} c_j .$$

The construction of strong constraints, which are consequences of inequality system (5.2), offers several advantages.

First, it avoids the time consuming process of checking all the constraints each time the tests are applied in order to elicit that either there is no feasible solution (the present branch may be closed) or that there is some obligatory fixing of variables.

Secondly, good lower bounds may be obtained for the objective function value. These bounds may be used in both multi- and single-branch procedures. The application in the former is quite natural, as was described in Chapter 3. In single branch enumerations, however, the next free variable to be fixed may be chosen in such a way that the new problem, so obtained, should have a minimal lower bound.

And last but not least, it is much easier to solve one-constraint problems, and they might result in feasible or near feasible solutions for the original problem.

It is well-known from the theory of linear inequalities that when consequences of an inequality system such as (5.2) are desired, we have to look for inequalities of the form

$$\mathbf{a}^T\mathbf{x} \geqq b_0 \tag{5.4}$$

where

$$\mathbf{a}^T = \mathbf{u}^T\mathbf{A}, \quad b_0 = \mathbf{u}^T\mathbf{b} \tag{5.5}$$

with a vector $\mathbf{u} \geqq \mathbf{0}$, chosen arbitrarily. The trivial case $\mathbf{u} = \mathbf{0}$ is excluded. The inequality (5.4) so obtained is called a surrogate constraint or shortly, an *s*-constraint.

Let us now consider *Problem (ii/u)*:

$$\min \mathbf{c}^T\mathbf{x} \tag{5.6}$$

$$\mathbf{a}^T\mathbf{x} \geqq b_0 \tag{5.7}$$

$$x_j \in \{0, 1\} \qquad (j = 1, \ldots n) \tag{5.8}$$

where the coefficients of inequalities (5.7) are given by expression (5.5).

The following simple example shows that the good choice of vector $\mathbf{u}$ is important. Consider the inequalities

$$-3x_1 + 2x_2 + 4x_3 - x_4 \geqq 3$$

$$7x_1 + 5x_2 - 6x_3 + 4x_4 \geqq 6\,.$$

Let us denote the set of vectors with 0–1 components satisfying the first, the second and both inequalities by S_1, S_2 and S_{12} respectively. Then obviously

$$S_{12} = S_1 \cap S_2\,.$$

Let us construct, in addition, the linear combination with weights $\mathbf{u}^3 = (1, 1)$

$$4x_1 + 7x_2 - 2x_3 + 3x_4 \geqq 9\,.$$

The corresponding solution set is denoted by S_3. Now

$$S_3 \supset S_{12} = S_1 \cap S_2$$

for example, the vector $\mathbf{x}^1 = (1, 1, 0, 0) \in S_3$, but $\mathbf{x}^1 \notin S_{12}$. This means that $\mathbf{x}^1$ does not satisfy one of the original inequalities. As $\mathbf{x}^1 \notin S_1$, the weight of the first

inequality may be increased. Let us take, for example, the vector $\mathbf{u}^4 = (2, 1)$. The solution set of the corresponding inequality

$$x_1 + 9x_2 + 2x_3 + 2x_4 \geqq 12$$

is denoted by S_4. Then $\mathbf{x}^1 \notin S_4$ and, in fact,

$$S_3 \supset S_4 \supset S_{12}$$

TABLE 5.1

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	u
x_1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	
x_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	
x_3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	
x_4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	
S_{12}															+		
S_1			+	+			+	+							+		(1, 0)
S_2						+			+	+			+	+	+	+	(0, 1)
S_3						+							+	+	+	+	(1, 1)
S_4														+	+	+	(2, 1)

as can be seen from Table 5.1, which contains the sets S_1, S_2, S_{12}, S_3, S_4. The + sign in row S_k and in the column of $\mathbf{x}^i$ means that

$$\mathbf{x}^i \in S_k .$$

It would seem natural to introduce the definition of strength for s-contraints in the following way. The s-contraint corresponding to the weight $\mathbf{u}^1$ is called stronger than that corresponding to multiplier vector $\mathbf{u}^2$, if the relation

$$S(\mathbf{u}^1) \subset S(\mathbf{u}^2)$$

holds for the corresponding solution set S. In our example vector $\mathbf{u}^4$ would result in a stronger s-constraint than vector $\mathbf{u}^3$. More practical definitions of s-constraint strength will be introduced later. In the above example, it is not possible to construct an s-constraint with less than three solutions. In general, we cannot expect to describe an inequality system by any single inequality, but most probably inequalities stronger than the ones originally given in system (5.2) can be constructed.

Now, let us return to Problem (ii/u). As inequality (5.4) is a consequence of inequality system (5.2), the following lemma obviously holds true.

Lemma 5.1: *Let us denote the optimal value of the objective function of Problem* (i) *by* z^* *and the optimal solution of Problem* (ii/u^0) *for a given vector* $\mathbf{u}^0$ *by* $\mathbf{x}^0$. *Then*

1° $z^* \geqq \mathbf{c}^T\mathbf{x}^0$

2° *If* $\mathbf{x}^0$ *satisfies constraints* (5.2), *then* $\mathbf{x}^0$ *is also an optimal solution for Problem* (i). This lemma suggests the following definition.

Definition 5.1: (Glover). *The s-constraint corresponding to the multiplier vector* $\mathbf{u}^1$ *is said to be stronger than that of vector* $\mathbf{u}^2$, *if*

$$\mathbf{c}^T\mathbf{x}^1 > \mathbf{c}^T\mathbf{x}^2,$$

where $\mathbf{x}^1$ *and* $\mathbf{x}^2$ *are optimal solutions of problems* (ii/$\mathbf{u}^1$) *and* (ii/$\mathbf{u}^2$) *respectively.*

At first, it might seem somewhat surprising that the problem with the worse optimum contains the better (stronger) *s*-constraint, but it provides a better estimation of the feasible set for Problem (i) – at least in the vicinity of the optimal solution. Namely, according to Lemma 5.1

$$z^* \geqq \mathbf{c}^T\mathbf{x}^1 > \mathbf{c}^T\mathbf{x}^2,$$

that is, the optimal solution of Problem (ii/$\mathbf{u}^1$) is closer to optimal solution (i), than that of Problem (ii/$\mathbf{u}^2$). If inequality system (5.2) contains only two constraints

$$\mathbf{a}^1\mathbf{x} \geqq b_1 \tag{5.9}$$

$$\mathbf{a}^2\mathbf{x} \geqq b_2, \tag{5.10}$$

where $\mathbf{a}^1$ and $\mathbf{a}^2$ denote row vectors, and the optimal solution $\mathbf{x}^1$ of Problem (ii/u) containing the *s*-constraint

$$(u_1\mathbf{a}^1 + u_2\mathbf{a}^2)\mathbf{x} \geqq u_1b_1 + u_2b_2 \tag{5.11}$$

does not satisfy constraints (5.9) and (5.10), then a new multiplier vector $\hat{\mathbf{u}}$ may be determined in such a way that the new *s*-constraint is not satisfied by $\mathbf{x}^1$

$$(\hat{u}_1\mathbf{a}^1 + \hat{u}_2\mathbf{a}^2)\mathbf{x}^1 < \hat{u}_1b_1 + \hat{u}_2b_2\,. \tag{5.12}$$

Vector $\hat{\mathbf{x}}$ should satisfy one of the constraints (5.9) and (5.10) because in the opposite case it could not even be feasible for Problem (ii/*u*). Let us suppose for simplicity that this is constraint (5.9). The desired vector $\hat{\mathbf{u}}$ is determined by

Lemma 5.2: *The vector* $\mathbf{x}^1$ *for which*

$$\mathbf{a}^1\mathbf{x}^1 > b_1 \qquad \text{and} \qquad \mathbf{a}^2\mathbf{x}^1 < b_2$$

satisfy inequality (5.11), *if and only if either* $u_2 = 0$ or $u_2 > 0$ *and*

$$\frac{u_1}{u_2} \geqq \frac{b_2 - \mathbf{a}^2\mathbf{x}^1}{\mathbf{a}^1\mathbf{x}^1 - b_1}\,. \tag{5.13}$$

Proof: For $u_2 = 0$, $\mathbf{x}^1$ satisfies the remaining inequality by supposition. If $u_2 > 0$, then (5.13) may be obtained by substituting vector $\mathbf{x}^1$ in inequality (5.11) and arranging it properly.

This simple lemma provides new multiplier vectors $\hat{\mathbf{u}}$, that is all vectors not satisfying inequality (5.13) and having $u_2 > 0$ may be used for this purpose.

This result may easily be generalized for the several-constraint case. Let us denote again an optimal solution of Problem (ii/**u**) by $\mathbf{x}^1$ and suppose that

$$\mathbf{a}^i\mathbf{x}^1 \geqq b_i \qquad i \in I$$

$$\mathbf{a}^i\mathbf{x}^1 < b_i \qquad i \notin I \,(l \leqq i \leqq m),$$

where $\mathbf{a}^i$ denotes row i of matrix **A**. Then the role of coefficient vectors $\mathbf{a}^1$ and $\mathbf{a}^2$ in Lemma 5.2 will be played by the vectors

$$\sum_{i \in I} u_i\mathbf{a}^i \quad \text{and} \quad \sum_{\substack{i \notin I \\ 1 \leqq i \leqq m}} u_i\mathbf{a}^i \text{ respectively,}$$

and the right hand sides b_1 and b_2 are substituted by

$$\sum_{i \in I} u_i b_i \quad \text{and} \quad \sum_{\substack{i \notin I \\ 1 \leqq i \leqq m}} u_i b_i \text{ respectively.}$$

5.3 A heuristic procedure for finding strong *s*-constraints

Let us define a series of problems, Problems (ii/$\mathbf{u}^k$), starting from any $\mathbf{u}^1 > 0$, for example $\mathbf{u}^1 = (1, 1, \ldots, 1)$. If the optimal solution of *Problem (ii/$\mathbf{u}^k$)* satisfies constraints (5.2), i.e. it is a feasible solution for Problem (i), then it is also optimal for Problem (i), by Lemma 5.1, and the procedure is finished. In the opposite case, define the set

$$I_k = \{i \mid u_i^k \mathbf{a}^i\mathbf{x}^k \geqq b_i\} \tag{5.14}$$

then the new multipliers are defined as follows

$$u_i^{k+1} = u_i^k \qquad i \notin I_k \tag{5.15}$$

$$u_i^{k+1} = u_i^k(r_k + \varepsilon) \quad i \in I - I_k, \tag{5.16}$$

where

$$I = \{1, 2, \ldots, m\}$$

and $\varepsilon > 0$, otherwise an arbitrary number. The value of ε should be as small as permitted by rounding errors, in order to change the *s*-constraints as little as possible, so that vector $\mathbf{x}_k$ can be excluded.

This is to avoid alternate entering and leaving of vectors to and from the solution sets of Problem (ii/$\mathbf{u}_k$). The coefficient r_k is defined by Lemma 5.2 as

$$r_k = \frac{\sum_{i \in I_k} u_i^k (b_i - \mathbf{a}^i \mathbf{x}^k)}{\sum_{i \in I - I_k} u_i^k (\mathbf{a}^i \mathbf{x}^k - b_i)} . \tag{5.17}$$

The procedure may be continued as long as the new s-constraint is stronger than the previous one

$$\mathbf{c}^T \mathbf{x}^1 < \mathbf{c}^T \mathbf{x}^2 < \ldots < \mathbf{c}^T \mathbf{x}^k \geqq \mathbf{c}^T \mathbf{x}^{k+1} . \tag{5.18}$$

The procedure is obviously finite. There is no guarantee, however, that the strongest s-constraint – according to definition (5.1) – may be found.

Example: Let us determine a strong s-constraint for the following problem

$$\begin{aligned} \min\ & (4x_1 + 2x_2 + 9x_3 + 3x_4 + 14x_5) \\ & x_1 + 5x_2 + 10x_3 + 10x_4 + 15x_5 \geqq 16 \\ & 5x_1 + 2x_2 + 10x_3 + x_4 + 12x_5 \geqq 12 \\ & 10x_1 + 3x_2 + 2x_3 + 5x_4 + x_5 \geqq 8 \\ & x_j \in \{0, 1\} \qquad (j = 1, \ldots, 5) . \end{aligned}$$

The s-constraint for multiplier vector $\mathbf{u}^1 = (1, 1, 1)$ is calculated first

$$16x_1 + 10x_2 + 22x_3 + 16x_4 + 28x_5 \geqq 36 .$$

An optimal solution of Problems (ii/$\mathbf{u}^k$) may be obtained by using any of the methods discussed in Chapters 2–4 for the knapsack problem. It is easily seen that the solution of the above Problem (ii/u^1) is,

$$\mathbf{x}^1 = (1, 1, 0, 1, 0), \quad \mathbf{c}^T \mathbf{x}^1 = 9, \quad I_1 = \{1, 3\} \neq I.$$

$$r_1 = \frac{0 + 10}{4} = 2.5$$

and let us use $\varepsilon = 0.1$. Then, according to expressions (5.15)–(5.16),

$$\mathbf{u}^2 = (1, 2.6, 1),$$

which results in the s-constraint

$$24x_1 + 13.2x_2 + 38x_3 + 17.6x_4 + 47.2x_5 \geqq 55.2 .$$

The optimal solution of Problem (ii/$\mathbf{u}^2$) is

$$x^2 = (0, 0, 1, 1, 0), \quad \mathbf{c}^T \mathbf{x}^2 = 12 > \mathbf{c}^T \mathbf{x}^1, \quad I_2 = \{1\} \neq I.$$

Similarly to the first iteration

$$r_2 = \frac{4}{2.6 + 1}, \quad \mathbf{u}^3 = (1, 3.1, 1.2).$$

The corresponding s-constraint is

$$28.5x_1 + 14.8x_2 + 43.4x_3 + 19.1x_4 + 53.4x_5 \geqq 62.8$$

with the optimal solution

$$\mathbf{x}^3 = (1, 0, 1, 0, 0), \quad \mathbf{c}^T\mathbf{x}^3 = 13 > \mathbf{c}^T\mathbf{x}^2, \quad I_3 = \{2, 3\} \neq I.$$

Continuing the procedure

$$r_3 = \frac{3 + 4}{4} = \frac{7}{4}, \quad \mathbf{u}^4 = (1.9, 3.1, 1.2).$$

The new s-constraint is

$$30.4x_1 + 24.4x_2 + 62.4x_3 + 38.1x_4 + 81.9x_5 \geqq 93.2$$

$$\mathbf{x}^4 + (0, 0, 1, 1, 0) \quad \mathbf{c}^T\mathbf{x}^4 = 12 < \mathbf{c}^T\mathbf{x}^3.$$

The procedure is stopped and the previous s-constraint

$$28.5x_1 + 14.8x_2 + 43.4x_3 + 19.1x_4 + 53.4x_5 \geqq 62.8$$

is accepted as the best one generated. In the next section, procedures will be given for generating an optimal s-constraint.

5.4 Another definition of s-constraint strength

Let us consider the following three problems (using the notation introduced in Section 5.2)

$$L_1 = \max_{\mathbf{u} \geqq \mathbf{0}} \min_{\mathbf{x} \in s_1(\mathbf{u})} \mathbf{c}^T\mathbf{x}, \text{ where} \tag{5.19}$$

$$s_1(\mathbf{u}) = \{\mathbf{x} \mid \mathbf{u}^T\mathbf{A}\mathbf{x} \geqq \mathbf{u}^T\mathbf{b}; \quad x_j \in \{0, 1\} \quad (j = 1, \ldots, n)\}$$

$$L_2 = \max_{\mathbf{u} \geqq \mathbf{0}} \min_{\mathbf{x} \in s_2(\mathbf{u})} \mathbf{c}^T\mathbf{x}, \text{ where} \tag{5.20}$$

$$s_2(\mathbf{u}) = \{\mathbf{x} \mid \mathbf{u}^T\mathbf{A}\mathbf{x} \geqq \mathbf{u}^T\mathbf{b}; \quad O \leqq x_j \leqq 1 \quad (j = 1, \ldots, n)\}$$

$$L_3 = \min_{\mathbf{u} \geqq \mathbf{0}} \max_{\mathbf{x}} \{\mathbf{u}^T(\mathbf{A}\mathbf{x} - \mathbf{b}) + \bar{z} - \mathbf{c}^T\mathbf{x} \mid x_j \in \{0, 1\} \quad (j = 1, \ldots, n)\} \tag{5.21}$$

where $\bar{z}$ is the best objective function value obtained so far.

If $s_1(\mathbf{u}) \neq \emptyset$, $s_2(\mathbf{u}) \neq \emptyset$ and $L_3 > -\infty$ respectively, then the corresponding problems (5.19), (5.20) and (5.21) have optimal solutions and they are denoted by $(\mathbf{u}^1, \mathbf{x}^1)$, $(\mathbf{u}^2, \mathbf{x}^2)$ and $(\mathbf{u}^3, \mathbf{x}^3)$ respectively. These problems determine three multiplier vectors $\mathbf{u}^1$, $\mathbf{u}^2$ and $\mathbf{u}^3$, thus they are also defining strongest s-constraints in three different senses. The s-constraints corresponding to problems (5.19) and (5.20) are of the form

$$\mathbf{u}^T\mathbf{A}\mathbf{x} \geqq \mathbf{u}^T\mathbf{b},$$

where $\mathbf{u} = \mathbf{u}^1$ and $\mathbf{u}^2$ respectively. The s-constraint obtained by solving (5.21) takes the form

$$\mathbf{u}^{3T}(\mathbf{A}\mathbf{x} - \mathbf{b}) + \bar{z} - \mathbf{c}^T\mathbf{x} - 1 \geqq 0\,.$$

Problem (5.19) results in Glover's definition of strongest surrogate constraint. There is no efficient exact method so far for solving this problem. A heuristic algorithm was discussed in Section 5.2.

Problem (5.20) results in the strongest s-constraint definition of Balas (1967). The set $s_2(\mathbf{u})$ is obtained from set $s_1(\mathbf{u})$ by relaxing the integer constraints $x_j \in \{0, 1\}$ to $0 \leqq x_j \leqq 1$. Therefore $L_1 \leqq L_2$, in other words Balas' definition may result in a somewhat weaker s-constraint but it has the advantage that (5.20) may be solved by linear programming. This transformation, however, is not trivial. It is discussed below. The reason for forming (5.20) is the same as that of (5.19), as was described in Section 5.1, that is, for each multiplier vector $\mathbf{u} \geqq \mathbf{0}$ the continuous version of Problem (ii/u) is formed and solved

$$\begin{aligned} &\min \mathbf{c}^T\mathbf{x} \\ &\mathbf{u}^T\mathbf{A}\mathbf{x} \geqq \mathbf{u}^T\mathbf{b} \qquad (5.22) \\ &0 \leqq x_j \leqq 1 \quad (j = 1, \ldots, n)\,. \end{aligned}$$

The multiplier vector $\mathbf{u}$, resulting in the worst optimum (the one closest to the optimum of Problem (i)) is accepted and the corresponding s-constraint is considered as the strongest one. This approach, however, is only of theoretical value because we should need to solve an infinite number of problems, one for each $\mathbf{u} \geqq \mathbf{0}$.

Let us consider the continuous version of Problem (i) as a primal problem, *Problem (P)* instead

$$\begin{aligned} &\min \mathbf{c}^T\mathbf{x} \\ &\mathbf{A}\mathbf{x} \geqq \mathbf{b} \qquad (P) \\ &0 \leqq x_j \leqq 1 \quad (j = 1, \ldots, n)\,. \end{aligned}$$

The dual *Problem (D)* of this problem may be written as follows

$$\begin{aligned} &\max (\mathbf{u}^T\mathbf{b} - \mathbf{t}^T\mathbf{e}) \\ &\mathbf{u}^T\mathbf{A} - \mathbf{t} \leqq \mathbf{c} \qquad (D) \\ &\mathbf{u} \geqq \mathbf{0}, \quad \mathbf{t} \geqq \mathbf{0}, \end{aligned}$$

where $\mathbf{e}$ is the vector of unit elements. Let us suppose that Problem (P) has a feasible solution and denote the optimal solutions of Problems (P) and (D) by $\hat{\mathbf{x}}$ and $(\hat{\mathbf{u}}, \hat{\mathbf{t}})$ respectively. The multipliers $\mathbf{u}$ of the strongest s-constraint in the sense of (5.20) may be determined with the help of the following statement:

Lemma 5.3: *There exists an optimal solution of form* $(\hat{\mathbf{u}}, \hat{\mathbf{x}})$ *for* (5.20) *where* $(\hat{\mathbf{u}}, \hat{\mathbf{t}})$ *is an optimal solution of Problem (D).*

Proof: First of all, let us denote an optimal solution of the problem

$$\min \{\mathbf{c}^T\mathbf{x} \mid \mathbf{u}^T\mathbf{A}\mathbf{x} \geqq \mathbf{u}^T\mathbf{b}; \quad 0 \leqq x_j \leqq 1 \ (j = 1, \ldots, n)\} \tag{5.23}$$

by $\mathbf{x}(\mathbf{u})$. As the constraints of (5.22) are consequences of those of Problem (P) for any $\mathbf{u} \geqq \mathbf{0}$, it is obvious that

$$\mathbf{c}^T\mathbf{x}(\mathbf{u}) \leqq \mathbf{c}^T\hat{\mathbf{x}} \quad \text{for any} \quad \mathbf{u} \geqq \mathbf{0}. \tag{5.24}$$

Therefore it is sufficient to show that there exists a vector $\mathbf{u}$ for which the equality sign holds. Let us consider for this purpose the case $\mathbf{u} = \hat{\mathbf{u}}$

$$z_1 = \mathbf{c}^T(\hat{\mathbf{u}}) = \{\min \mathbf{c}^T\mathbf{x} \mid \hat{\mathbf{u}}^T\mathbf{A}\mathbf{x} \geqq \hat{\mathbf{u}}^T\mathbf{b}; \quad 0 \leqq x_j \leqq 1 \ (j = 1, \ldots, n)\}. \tag{5.25}$$

The dual of this problem is the following:

$$w_1 = \max \{\hat{\mathbf{u}}^T\mathbf{b}s - \mathbf{t}^T\mathbf{e} \mid \mathbf{A}^T\hat{\mathbf{u}}s - \mathbf{t} \leqq \mathbf{c}, \quad s \geqq 0, \quad \mathbf{t} \geqq \mathbf{0}\} \tag{5.26}$$

where

$$\mathbf{e}^T = (1, 1, \ldots, 1).$$

The duality theorem of linear programming states that $z_1 = w_1$. On the other hand,

$$w_1 \geqq \hat{\mathbf{u}}^T\mathbf{b} - \hat{\mathbf{t}}^T\mathbf{e}, \tag{5.27}$$

as $s = 1$, $\mathbf{t} = \hat{\mathbf{t}}$ is a feasible solution for (5.26). But the right hand side of (5.27) is the optimum of Problem (D), hence

$$\mathbf{c}^T\mathbf{x}(\hat{\mathbf{u}}) = z_1 = w_1 \geqq \hat{\mathbf{u}}^T\mathbf{b} - \hat{\mathbf{t}}^T\mathbf{e} = \mathbf{c}^T\mathbf{x}. \tag{5.28}$$

Inequalities (5.24) and (5.28) together prove the lemma with $\hat{\mathbf{x}} = \mathbf{x}(\hat{\mathbf{u}})$.

We have obtained the simple result, namely that the optimal values of the dual variables provide optimal multipliers $\mathbf{u} \geqq \mathbf{0}$, in the sense defined by (5.20).

Let us now turn to problem (5.21). This definition of strongest surrogate constraint was given by Geoffrion (1969). In our problem, $\bar{z}$ denotes the objective function value of the best solution found so far. When no feasible solution has yet been found, then $\bar{z}$ is an upper bound of the objective function, for example

$$\bar{z} = 1 + \sum_{j=1}^{n} c_j.$$

This choice does not exclude any solution by the objective function constraint

$$-\mathbf{c}^T\mathbf{x} \geqq -\bar{z} + 1\,. \tag{5.29}$$

If there exists a vector $\mathbf{u} \geqq \mathbf{0}$, such that

$$\max_{\mathbf{x}} \{\mathbf{u}^T(\mathbf{A}\mathbf{x} - \mathbf{b}) + \bar{z} - \mathbf{c}^T\mathbf{x} \mid x_j \in \{0, 1\} \quad (j = 1, \ldots, n)\} \leqq 0, \tag{5.30}$$

then obviously Problem (i), which is repeated here for convenience (including the objective function constraint)

$$\begin{gathered} \min \mathbf{c}^T\mathbf{x} \\ -\mathbf{c}^T\mathbf{x} \geqq -\bar{z} + 1 \\ \mathbf{A}\mathbf{x} \geqq \mathbf{b} \\ x_j \in \{0, 1\} \quad (j = 1, \ldots, n) \end{gathered} \tag{5.31}$$

has no feasible solution. Therefore the purpose of (5.21) is either to find a vector $\mathbf{u} \geqq \mathbf{0}$, for which inequality (5.30) holds, or to find one closest to this binary infeasibility, i.e. a strongest s-constraint in the sense of Geoffrion.

Let us now return to the method for solving (5.21). Obviously, the following three quantities are equal to each other

$$\min_{\mathbf{u} \geqq \mathbf{0}} \max_{\mathbf{x}} \{\mathbf{u}^T(\mathbf{A}\mathbf{x} - \mathbf{b}) + \bar{z} - \mathbf{c}^T\mathbf{x} \mid x_j \in \{0, 1\} \quad (j = 1, \ldots, n)\} \tag{5.32}$$

$$\min_{\mathbf{u} \geqq \mathbf{0}} \max_{\mathbf{x}} \{(\mathbf{u}^T\mathbf{A} - \mathbf{c}^T)(\mathbf{x} - \mathbf{u}^T\mathbf{b}) + \bar{z} \mid 0 \leqq x_j \geqq 1 \quad (j = 1, \ldots, n)\} \tag{5.33}$$

$$\min_{\mathbf{u} \geqq \mathbf{0}} \min_{\mathbf{t} \geqq \mathbf{0}} \{\mathbf{t}^T\mathbf{e} + \bar{z} - \mathbf{u}^T\mathbf{b} \mid \mathbf{t} \geqq \mathbf{u}^T\mathbf{A} - \mathbf{c}^T\} \tag{5.34}$$

where $\mathbf{e}$ is an n dimensional vector all components of which are 1. The first equality follows from the fact that the variables of the objective function of (5.32) are to be maximized independently from each other. The second equality is a consequence of the duality theorem of linear programming. The third problem may be solved by linear programming. If, during the process of solving this problem, the objective function reaches zero or goes below zero, then $L_3 \leqq 0$ and there is no feasible solution of (5.31). In the opposite case, if the optimum of the problem, $L_3 > 0$, then let us denote the optimal solution of (5.34) by $(\hat{\mathbf{u}}, \hat{\mathbf{t}})$. In this case

$$\begin{aligned} 0 < L_3 = \max_{\mathbf{x}} \{\hat{\mathbf{u}}^T(\mathbf{A}\mathbf{x} - \mathbf{b}) + \bar{z} - \mathbf{c}^T\mathbf{x} \mid 0 \leqq x_j \leqq 1 \; (j = 1, \ldots, n)\} \leqq \\ \leqq \max_{\mathbf{x}} \{\mathbf{u}^T(\mathbf{A}\mathbf{x} - \mathbf{b}) + \bar{z} - \mathbf{c}^T\mathbf{x} \mid 0 \leqq x_j \leqq 1 \; (j = 1, \ldots, n)\} \end{aligned} \tag{5.35}$$

for any $\mathbf{u} \geqq \mathbf{0}$.

The above inequality implies that the optimal solution of the inequality of (5.35), i.e. the vector $\mathbf{x}$ of dual optimal variables of (5.34) satisfies the constraints of (5.31),

apart from integrality restrictions. If the latter are also satisfied, then a new feasible solution has been found, the value of $\bar{z}$ is updated and the process is repeated. In the opposite case we have obtained the strongest s-constraint in the third sense.

It is easy to see that a multiplier vector $\hat{\mathbf{u}}$ solves (5.20) if and only if it is also optimal for (5.21). In spite of this fact, the two s-constraint definitions (those of Balas and Geoffrion) are different because Geoffrion also mixes the objective function of the s-constraint with weight 1.

Very good results are reported in Geoffrion (1969). By applying his s-constraint concept, a speeding up by a factor of about 100 is obtained in the enumeration method. It is also conjectured that the exponential growth in computer time, when the number of variables is increased, may be reduced to a low-order polynomial growth.

5.5 Tests

A quite general schema of enumeration algorithms was presented in Chapter 2. The determination of consequences (obligatory variable fixing) and of empty pseudo solutions (possibility of backtracking) in Steps 2 and 3, respectively, were not detailed. This is the purpose of the tests discussed in this section, and partly in the next chapter.

Let us consider again the usual problem

$$\min \mathbf{c}^T\mathbf{x} \tag{5.36}$$

$$\mathbf{a}^i\mathbf{x} \geqq b_i \quad (i = 0, 1, \ldots, m) \tag{5.37}$$

$$x_j \in \{0, 1\} \quad (j = 1, 2, \ldots, m), \tag{5.38}$$

where a^i denotes row i of matrix $\mathbf{A}$

$$\mathbf{a}^i = (a_{i1}, a_{i2}, \ldots, a_{in}),$$

and row 0 is the objective function constraint:

$$-\mathbf{c}^T\mathbf{x} \geqq -\bar{z} + 1,$$

denoting by $\bar{z}$ the best objective function value obtained so far.

We can suppose that one or several of the s-constraints are also included in inequalities (5.37). As before, all the coefficients are supposed as being integer and $\mathbf{c} \geqq \mathbf{0}$.

The tests will be formulated for the starting state of the problem but they can still be applied in the same way if some of the variables are already fixed. In the latter case the transformed problem is used (obtained by inserting the fixed values of variables and by rearranging it to the same form.) Then, obviously, the "no solution" result should be understood as "no solution in the present pseudo-solution", i.e. the value of at least one of the fixed variables should be changed, otherwise no solution may be obtained.

Notation

L, U lower and upper bounds, resp., for the number of (free) variables which must take the value 1 in order to obtain a feasible solution.

$\overline{\sum_j}_M a_{ij}$ the sum of the largest M number among $a_{i1}, a_{i2}, \ldots, a_{in}$.

$\overline{\sum_j}_{M,N} a_{ij}$ the largest possible sum taken from the elements $a_{i1}, a_{i2}, \ldots, a_{in}$ by choosing at least M and at most N elements.

$\underline{\sum_j}_M a_{ij}, \underline{\sum_j}_{M,N} a_{ij}$ the same as the corresponding $\overline{\sum}$, only the word "largest" is substituted by the word "smallest".

The proper determination of numbers L and U is important for several reasons. First of all, because they are often calculated, simple methods should be used. On the other hand, better lower and upper bounds may substantially speed up the algorithm when the number of free variables is still large. As a consequence, several methods of different complexity are desired for calculating the bounds L and U:

1° Determine for each $i = 0, 1, \ldots, m$, L_i and U_i the minimum and the maximum of the number of variables which may take the value 1 and still satisfy constraint i

$$\overline{\sum_j}_{L_i - 1} a_{ij} < b_i \leqq \overline{\sum_j}_{L_i} a_{ij}.$$

and

$$\overline{\sum_j}_{U_i} a_{ij} \geqq b_i > \overline{\sum_j}_{U_i + 1} a_{ij}.$$

The largest of these lower bounds and the smallest of these upper bounds usually provide good bounds for the entire problem

$$L = \max_i L_i, \qquad U = \min_i U_i$$

2° A faster calculation is obtained if only one or only a few constraints is/are taken into consideration in the method described in 1°. Then preferably strong constraints (e.g. the s-constraints) are to be used.

3° Minimize (maximize)

$$\sum_j x_j$$

$$\mathbf{u}^T \mathbf{A} \mathbf{x} \geqq \mathbf{u}^T \mathbf{b}$$

$$x_j \in \{0, 1\} \quad (j = 1, \ldots, n)$$

where $\mathbf{u} \geq \mathbf{0}$ is an arbitrary vector (an s-constraint is determined). Obviously these two problems will result in a lower (upper) bound L and U, respectively.

4° Minimize (maximize)

$$\sum_j x_j$$

$$\mathbf{Ax} \geqq \mathbf{b}$$

$$0 \leqq x_j \leqq 1 \quad (j = 1, \ldots, n)$$

and the desired lower and upper bounds L and U are also obtained, taking the integer parts of the optima

$$L = -[-\Sigma x_j^1], \quad U = [\Sigma x_j^2],$$

where $\mathbf{x}^1$ ($\mathbf{x}^2$) is the optimal solution of the minimization (maximization) problem.

Further methods for determining L and U will be discussed in the next chapter. Throughout the algorithm, a good compromise could be made bearing in mind the computer time used and the accuracy of the obtained bounds. The frequency of the calculations is also important.

Now several feasibility and consequence tests will be discussed. Both kinds of tests are understood in such a way that each time a new feasible solution is found, the objective function constraint is adjusted and thus only better solutions are sought.

Test 1: (a) There is no solution in the present branch if

$$L > U.$$

(b) There is at most one solution in the present branch if either

$$\text{(b1)} \quad L = U = 0$$

$$\text{(b2)} \quad L = U = f,$$

where f is the number of free variables ($f = n$ at the beginning).

Note: when determining L and U, infeasibility may already be obvious. Then, using the symbols $L = +\infty$ and $U = -\infty$ respectively (depending on where the infeasibility was found). *Test* 1(a) also summarizes these cases. In case (b) all free variables are set equal to 0 (to 1 in the second case) and the solution so obtained is checked for feasibility. (The fixings are obligatory!) If the answer is positive, a new feasible solution is found and the objective function constraint is adjusted. In any case, backtracking takes place. In the following tests, F denotes the set of subscripts of free variables and $\bar{b}_i$ stands for the transformed right hand side

$$\bar{b}_i = b_i - \sum_{j \notin F} a_{ij}\delta_j,$$

where δ_j is the value of x_j in the present branch.

Test 2: If

$$\overline{\sum_{j \in F}}_{L,U} a_{ij} < \bar{b}_i \tag{5.39}$$

for at least one of the subscripts $i = 0, 1, \ldots, m$, then there is no solution in the present branch; backtracking is applied.

Test 3: If the inequality

$$a_{ik} > \overline{\sum_{j \in F}}_{L+1,\, U+1} a_{ij} - \bar{b}_i \tag{5.40}$$

holds for an i $(0 \leqq i \leqq m)$, where a_{ik} is a member of the sum

$$\overline{\sum_{j \in F}}_{L,\, U} a_{ij}, \tag{5.41}$$

then the fixing $x_k = 1$ is obligatory.

As a justification of the test, condition (5.40) may be reformulated as

$$\overline{\sum_{\substack{j \in F \\ j \neq k}}}_{L,\, U} a_{ij} < \bar{b}_i$$

which shows that the fixing $x_k = 0$ would give an infeasible branch. Analogously,

Test 4: If the inequality

$$a_{ik} < \bar{b}_i - \overline{\sum_{j \in F}}_{L-1,\, U-1} a_{ij} \tag{5.42}$$

holds for an i $(0 \leqq i \leqq m)$, where a_{ik} is not a member of sum (5.41) and $k \in F$, then the fixing $x_k = 0$ is obligatory.

In the opposite case, if $x_k = 1$, at least $L - 1$, at most $U - 1$ further free variables should take the value 1. Then inequality (5.42) is reformulated as

$$a_{ik} + \overline{\sum_{j \in F}}_{L-1,\, F-1} a_{ij} < \bar{b}_i,$$

which clearly shows that no solution exists in the present branch, i.e. the fixing $x_k = 0$ is really obligatory.

The following three tests may be applied only if at least one feasible solution of the problem has already been found.

Let us consider the following problem

$$\begin{gathered} \min z = \sum_{j \notin F} c_j \delta_j + \sum_{j \in F} c_j x_j \\ \sum_{i=0}^{m} \sum_{j \in F} u_i a_{ij} x_j \geqq \sum_{i=0}^{m} u_i \bar{b}_i, \\ 0 \leqq x_j \leqq 1 \quad (j = 1, 2, \ldots, n) \end{gathered} \tag{5.43}$$

where the multipliers $u \geqq 0$ $(u \neq 0)$ are determined as described in Section 5.3. The optimal solution of (5.43) which may easily be obtained (see Lemma 1.2), is obviously a lower bound for our problem. The optimum of (5.43) is denoted by $\hat{z}$.

Furthermore, the optima of the modified problems that are obtained by substituting $x_k = 0$ and 1 respectively to (5.43), are denoted by z_k and z^k respectively. Only the free variables $x_k (k \in F)$ are considered. Then the following tests may be used:

Test 5: If the inequality

$$\hat{z} > \bar{b}_0$$

holds, then there is no solution in the present branch.

Test 6: The inequality

$$z_k > \bar{b}_0$$

implies that the fixing $x_k = 1$ is obligatory.

Test 7: The inequality

$$z^k > \bar{b}_0$$

implies that the fixing $x_k = 0$ is obligatory.

When determining bounds L and U, some of the constraints may obviously become superfluous. Should this occur, these constraints may be temporarily deleted until the next backtracking. This is the purpose of the following statement.

Test 8: If the inequality

$$\sum_{j \in F}{}_{L,U}\, a_{ij} \geqq \bar{b}_i \tag{5.44}$$

holds, then constraint i may be deleted and the corresponding multiplier u_i is set to 0. Inequality (5.44) means that any solution in the present branch satisfies constraint i, as only solutions with the property

$$L \leqq \sum_{j \in F} x_j \leqq U$$

are considered, according to the definition of bounds L and U.

Tests 1–8 are mainly those of Glover (1965). There are many other tests in the literature. A good survey may be found, for example, in Zoutendijk (1970). Substantially different kinds of tests are those using more than one constraint at a time. As an example, let us consider constraint i which is at present *not* satisfied: $\bar{b}_i > 0$; and another constraint $h \neq i$. Let us define the sets

$$J_i^+ = \{j \mid j \in F, \quad a_{ij} > 0\},$$

where F is the subscript set of presently free variables and $\bar{b}_i$ is the transformed right hand side of the inequality. Then we can describe

Test 9: If the inequality

$$\overline{\sum_{j \in J_h^+}}^{k_i} a_{hj}^- + \sum_{j \in J_h^+} a_{hj} < b_h \tag{5.45}$$

holds, then the present branch is infeasible. The number k_i is defined by the inequalities

$$\overline{\sum_{j \in F}}^{k_i - 1} a_{ij} < \bar{b}_i \leqq \overline{\sum_{j \in F}}^{k_i} a_{ij}, \tag{5.46}$$

and the minus sign in the superscript means the negative part

$$d^- = \begin{cases} d & \text{if } d < 0 \\ 0 & \text{otherwise.} \end{cases}$$

A number of further tests may be designed by substituting $x_k = 0$ (or $x_k = 1$) (for a free variable $k \in F$) to either the test inequality (5.45) or to the inequalities defining the number k_i. If the inequality (5.45) holds under the modified circumstances (but not without it), then the modification $x_k = 0$ (or $x_k = 1$) is prohibited, i.e. the variable fixing $x_k = 1$ (or $x_k = 0$) is obligatory. The design of further tests and the effective use of large-scale testing will be described in Chapter 11 in more detail.

5.6 Description of the algorithm

The algorithm is substantially the one described in Section 2.5. It is extended by the use of *s*-constraints and the tests stated above. Naturally the objective function constraint is used and adjusted after each new feasible solution found.

The notation and notions of Section 2.5 will be used. The fixings $x_j = 1$ and $x_j = 0$ will be denoted by the symbols j and $\bar{j}$ respectively. A variable is tied (in hand writing the corresponding term j or $\bar{j}$ is underlined) if the foregoing variable fixings already imply the present value of the corresponding variable, i.e. the opposite value results in an infeasible pseudo-solution. This may happen in two different ways. First of all this opposite value has already been considered and the corresponding branch is fathomed by the algorithm and a backtracking step resulted in the tying. In the second case one of the tests has shown the infeasibility of the branch. As in Chapter 2, ϕ denotes the ordered pseudo-solution carrying also the state of the enumeration. Furthermore the set of feasible solutions is denoted by S

$$S = \{\mathbf{x} \mid \mathbf{a}^i \mathbf{x} \geqq b_i \, (i = 0, 1, \ldots, m), x_j \in \{0, 1\} \, (j = 1, \ldots, n)\}. \tag{5.47}$$

It is clear from the defining equation (5.47) that the objective function constraint ($i = 0$) is included, i.e. set S is changed whenever a new feasible solution is found because then b_0 is adjusted to exclude all solutions with no smaller objective function value.

The algorithm for (5.36)–(5.38) may be summarized in the following steps.

Step 1: Let $\phi = (\emptyset)$, that is all variables are free,

$$F = \{1, 2, \ldots, n\}.$$

If the corresponding solution (the root of $\phi = (\emptyset)$)

$$\mathbf{x}^* = (0, \ldots, 0) \in S,$$

then this is also the optimal solution of the problem. In the opposite (and usual) case

Step 2: Determine the present value of the bounds L and U. (See Section 5.4).

Step 3: Apply Test 1. If condition (a) is satisfied, then go to Step 10. If condition (b) holds, then all free variables are set equal to 0 (Case (b) 1) or to 1 (Case (b) 2) and tied. Then go to Step 9. If none of the conditions is satisfied the next step is taken.

Step 4: Apply Test 2. If a positive result is obtained go to Step 10, otherwise to the next step.

Step 5: Apply Tests 3–7 in their original order. If the condition of Test 5 is satisfied, then go to Step 10, if any other test gives a positive result then the corresponding variable is fixed (and tied) and it sterm is added to the present pseudo-solution ϕ; Step 2 is repeated. If all the tests are tried without result, then:

Step 6: For all rows satisfying the condition of Test 8, the multiplier u_i in the s-constraint is set equal to 0. If there is a change of multiplier, Tests 3–4 are applied for the s-constraint only.

Step 7: If there are no more free variables, then go to Step 9. Otherwise if $\phi = (\emptyset)$ or there is no change since this step was visited last time, then fix one of the free variables at value 1 without tying it. Continue with the next step.

Step 8: Check the present pseudo-solution to determine whether its root (all free variables are temporarily set equal to 0) is feasible. If the answer is positive, the new solution is stored as the best one so far and denoted by $\mathbf{x}^*$, then modify the value b_0. If the ordered pseudo-solution ϕ has not changed since Step 2 was visited last time, go to Step 7, otherwise go to Step 2.

Step 9: The present pseudo-solution ϕ contains only one solution. If it is feasible, then it is registered as the best one so far, it is denoted by $\mathbf{x}^*$, the value of b_0 is adjusted and the next step is taken.

Step 10: (Backtracking.) If $\phi = (\emptyset)$ or all variables in ϕ are tied, then go to Step 12. Otherwise the last untied (not underlined) variable is set equal to its opposite value and tied, all variables following it in the ordered pseudo-solution ϕ are left out, i.e. they are considered as free variables.

Step 11: New multipliers u_i are calculated for all rows. Go to Step 2.

Step 12: The algorithm is finished. If the solution x* is feasible for (5.36)–(5.38), then it is also optimal. In the opposite case there is no feasible solution of the problem.

Note 1°: If there is no feasible solution available at the beginning, then the objective function constraint is dummy, thus it is not checked until the first feasible solution is found. Accordingly, tests 5–7 are used only if a feasible solution is already known.

Note 2°: Because the frequent calculation of bounds L and U, for example, after each fixing, would be too time consuming it is only done after several fixings using different methods, depending on the state of the enumeration. Between two recalculations an adjustment of the bounds is made; after each variable fixing at value 1, both L and U are decreased by 1.

Note 3°: The use of the tests may be advantageous in a different order. For example it takes less time to apply tests 2, 3 and 4 to one constraint and then to take the next constraint. On the other hand – especially if the present pseudo-solution is likely to contain a feasible solution – sometimes it is better to apply the tests only for a few constraints, mainly for the s-constraints or for some other momentarily "difficult" constraints.

Note 4°: The weight of the objective function constraint in the s-constraint may vary depending on how far we have progressed in the enumeration and what is the number of free variables.

Note 5°: From the point of view of finding better solutions sooner, the choice of the Step 7 free variables is critical. One idea may be to use the objective function and an s-constraint and evaluate the variables according to the ratios

$$\frac{c_{j_1}}{a_{j_1}} \leqq \frac{c_{j_2}}{a_{j_2}} \leqq \ldots \leqq \frac{c_{j_k}}{a_{j_k}} .$$

Naturally the variables with $a_j = 0$ are ranked last. There are many other possibilities as will be seen in the following chapters. Sometimes it might be useful to make several voluntary fixings at Step 7 instead of making only one. The finding of better solutions earlier is important for several reasons. First of all the better feasible solution we know the stronger objective function constraint we get and thus we can speed up the algorithm. On the other hand, if the run is terminated before the end of the algorithm it is essential to have as good solutions as possible.

A detailed discussion of how to construct an effective enumeration algorithm is left to Chapter 11.

Example: Solve the following numerical problem

$$\begin{aligned}
\min \quad & 3x_1 + 5x_2 + x_3 + 6x_4 + 2x_5 + 8x_6 + 10x_7 \\
& -3x_1 - 5x_2 - x_3 - 6x_4 - 2x_5 - 8x_6 - 10x_7 \geqq b_0 \\
& 5x_1 - 2x_2 + 7x_3 + x_4 + 4x_5 + 10x_6 - 5x_7 \geqq 20 \\
& 8x_1 + 7x_2 - 2x_3 + 5x_4 + 6x_5 - 4x_6 + 12x_7 \geqq 11 \\
& -7x_1 + 7x_2 - 5x_3 + 7x_4 - x_5 + 8x_6 + 4x_7 \geqq 9 \\
& x_j \in \{0, 1\} \qquad (j = 1, \ldots, 7) .
\end{aligned}$$

As we do not known any feasible solution at the beginning,

$$b_0 = -\sum_j c_j = -35 .$$

Using the procedure described in Section 5.2, an s-constraint may be determined:

$$5.4x_1 + 13.5x_2 + 0.6x_3 + 15.4x_4 + 9.9x_5 + 19.4x_6 + 10.7x_7 \geqq 48.7,$$

where the multiplier vector

$$\mathbf{u} = (0, 1.3, 1, 1.3)$$

was used. (For $u_0 = 0$ see note 1°.)

The steps of the algorithm are as follows:

Step 1: $\phi = (\emptyset)$, $\mathbf{x}^* = (0, 0, 0, 0, 0, 0\,0) \notin S$.

Steps 2: The first method for calculating bounds L, U is used.

$$\begin{aligned}
L_1 &= 3, & U_1 &= 7 \\
L_2 &= 1, & U_2 &= 7 \\
L_3 &= 2, & U_3 &= 7 \\
L_4 &= 4, & U_4 &= 7
\end{aligned}$$

L_0 and F_0 are not calculated as there is no feasible solution yet.
L_4 and F_4 are computed from the s-constraint.

$$L = \max_i L_i = 4 \qquad U = \min_i U_i = 7$$

Steps 3–4: Tests 1 and 2 have no results.

Step 5: Test 3, for constraint 1, holds

$$a_{1,6} > \sum_{j \in F_{4,7}} a_{ij} - b_1 \quad (10 > 27 - 20)$$

thus $x_6 = 1$ is an obligatory fixing

$$\phi = (\underline{6}).$$

Now the transformed right hand sides are calculated

$$\bar{b}_1 = 20 - 10 = 10$$

$$\bar{b}_2 = 11 + 4 = 15$$

$$\bar{b}_3 = 9 - 8 = 1$$

$$\bar{b}_4 = 48.7 - 19.4 = 29.3.$$

The set of free variables is also reduced

$$F = \{1, 2, 3, 4, 5, 7\}.$$

Step 2: The bounds are reduced by 1 (see Note 2°).

$$L = 3, \qquad U = 6.$$

Steps 4–8: There is no result from the tests.

Step 7: One of the free variables should be fixed at value 1 (see Note 5°) according to the ratios c_j/a_{4j}. The voluntary fixing $x_5 = 1$ is chosen, thus

$$\phi = (\underline{6}, 5).$$

The transformed right hand sides are

$$b_1 = 6, \quad b_2 = 9, \quad b_3 = 2, \quad b_4 = 19.4$$

Step 8: $x = (0, 0, 0, 0, 1, 1, 0) \notin S$.

Step 2: L and U reduced by 1. $L = 2$, $U = 5$.

Steps 3–6: No results from the tests.

Step 7: According to our choice rule (variable x_2 has the best ratio)

$$\phi = (\underline{6}, 5, 2).$$

The transformed right hand side

$$\bar{\mathbf{b}} = (8, 2, -5, 5.9).$$

Step 8: $x = (0, 1, 0, 0, 1, 1, 0) \notin S$.

Step 2: $L = 1$, $U = 4$.

Steps 3–4: Tests 1 and 2 have no results.

Step 5: Test 3 may be applied successfully to constraint 1 with the result that $x_3 = 1$ is obligatory, thus

$$\phi = (\underline{6}, 5, 2, \underline{3})$$

$$\bar{\mathbf{b}} = (1, 4, 0, 5.3).$$

Step 2: $L = 1$, $U = 3$. ($L = 1$ can always be written instead of $L = 0$ whenever there is at least one positive element among the components of the transformed right hand side vector $\bar{\mathbf{b}}$.

Steps 3–5. No results from the tests.

Step 6: Test 8 may be applied for row 2.
Test 3 may be applied for the new s-constraint

$$-2.6x_1 + 10.4x_4 - 1.3x_7 \geqq 1.3$$

resulting in the obligatory fixing $x_4 = 1$. Thus

$$\phi = (\underline{6}, 5, 2, \underline{3}, \underline{4}),$$

and constraint 2 may be removed temporarily (until the next backtracking).

Step 8: $x = (0, 1, 1, 1, 1, 1, 0) \in S$ Feasible solution!

$$\mathbf{x}^* = (0, 1, 1, 1, 1, 0) \quad b_0 = -21.$$

The adjusted objective function constraint naturally results in a backtracking. (This is a consequence of the fact that $\mathbf{c} \geqq \mathbf{0}$)

$$-3x_1 - 10x_7 \geqq 1.$$

Step 2: $L = L_0 = -\infty$

Step 10: Backtracking:

$$\phi = (\underline{6}, 5, \overline{2})$$

$$\bar{\mathbf{b}} = (-11, 6, 9, 2, 19.4) \quad \text{(including } b_0\text{!)}$$

Step 2: After the backtracking the bounds are recalculated

$$\begin{array}{llll} L_0 = 0 & U_0 = 3 & & \\ L_i = 1 & U_i = 4 & & (i = 1, 2, 3) \\ L_4 = 2 & U_4 = 4 & & \\ \hline L = 2 & U = 3 & & \end{array}$$

Steps 3–4: No results from the tests.

Step 5: Test 3 applied for $i = 4$ results in $x_4 = 1$

$$\phi = (\underline{6}, \overline{5}, \underline{2}, \underline{4})$$

$$\bar{\mathbf{b}} = (-5, 5, 4, -5, 4).$$

Step 2: $L = 1$, $U = 2$.

Steps 3–4: No results.

Step 5: Test 4 applied for $i = 0$ results in $x_7 = 0$

$$\phi = (\underline{6}, 5, \underline{\overline{2}}, \underline{4}, \underline{\overline{7}}).$$

The transformed right hand side vector is unchanged.

Step 2: $L = 1$, $U = 1$.

Steps 3:–4: No results

Step 5: Test 3 applied for $i = 3$ results in $x_1 = 1$, and because of $U = 1$, $x_3 = 0$

$$\phi = (\underline{6}, 5, \underline{\overline{2}}, \underline{4}, \underline{\overline{7}}\ \underline{1}, \underline{\overline{3}}).$$

Step 8: $(1, 0, 0, 1, 1, 1, 0) \notin S$.

Step 10:

$$\phi = (\underline{6}, \underline{\overline{5}})$$

$$\mathbf{b} = (-13, 10, 15, 1, 29.3).$$

Step 2: $L = 3$, $U = 3$ (from $L_4 = 3$ and $U_0 = 3$).

Steps 3–4: No results.

Step 5: Test 3 applied for $i = 1$ results in $x_3 = x_1 = 1$

$$\phi = (\underline{6}, \underline{\overline{5}}, \underline{3}, \underline{1})$$

$$\bar{\mathbf{b}} = (-9, -2, 9, -13, 23.3).$$

Step 2: $L = 2$, $U = 1$ ($L_3 = 2$, $U_0 = 1$).

Step 3: Test 1 shows the infeasibility of the present pseudo-solution.

Step 10: All variables are tied (underlined).

Step 12: End of algorithm.

Optimal solution: $\mathbf{x}^* = (0, 1, 1, 1, 1, 1, 0)$

Optimum: $z^* = 22$.

CHAPTER 6

A MODIFIED ADDITIVE ALGORITHM

6.1 Introduction

The method described in the present chapter is based on the work of Balas (1965) which was one of the earliest papers using the enumeration method in discrete programming. Balas' method has been extensively tested and modified by many people, e.g. by Fleischmann (1967), Freeman (1966), Glover and Zionts (1965), and by Petersen (1967). Balas himself has also developed his algorithm further but the new algorithm – called the filter method and which will be discussed in Chapter 8 – does not make the examination of the old one unnecessary. The filter method uses the simplex method to solve the continuous version of some discrete subproblems. On the other hand, the additive algorithm is a pure implicit enumeration method and may thus have some advantages, especially in the case of large, specially structured problems, when the solution of the continuous problem would take a long time.

While retaining the most important features (evaluation of free variables, set of candidate variables, etc.) the algorithm itself will be different from that of Balas. The purpose of changing the method is twofold. First of all, the algorithm will be presented in the framework described in Section 2.5 and will therefore be closer to the other parts of this book. Secondly, a great number of modifications and extensions become very easy to make including features used in Chapter 5 and those suggested in the above papers. Furthermore the memory requirement of the modified algorithm is substantially less than that of the original one.

6.2 Notation, notions

Let us consider the following problem

$$\min z = \mathbf{c}^T\mathbf{x} \tag{6.1}$$

$$\mathbf{A}\mathbf{x} - \mathbf{y} = \mathbf{b} \tag{6.2}$$

$$x_j \in \{0, 1\} \quad (j = 1, \ldots, n) \tag{6.3}$$

$$\mathbf{y} \geqq \mathbf{0}, \tag{6.4}$$

where $\mathbf{A}$ is a given $m \times n$ matrix, $\mathbf{c} \geqq \mathbf{0}$ and $\mathbf{b}$ are given n and m component vectors, respectively. $\mathbf{x}$ and $\mathbf{y}$ are n and m component variable vectors, respectively. As has already been seen in Chapter 1, all bounded linear discrete problems can be

formulated as in (6.1)–(6.4). An $(n + m)$ dimensional vector $\mathbf{u} = (\mathbf{x}, \mathbf{y})$ is said to be the solution and feasible solution if it satisfies the constraints (6.2)–(6.3) and (6.2)–(6.4) respectively. A feasible solution $\mathbf{u}$ is called optimal if it minimizes the objective function (6.1) among all feasible solutions. The objective function constraint is assumed to be included in equations (6.2) in the following form

$$-\mathbf{c}^T\mathbf{x} - \mathbf{y}_1 = -\varepsilon - z^*,$$

where $\mathbf{x}^*$ is the best feasible solution found so far and

$$z^* = \mathbf{c}^T\mathbf{x}^*,$$

and $\varepsilon > 0$ is a sufficiently small number.

If no feasible solution has yet been found, then

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

A solution may be represented by the set J_s of subscripts of the variables x_j taking the value. 1. Then the solution $(\mathbf{x}, \mathbf{y})$ itself:

$$x_j^s = \begin{cases} 1 & \text{if } j \in J_s \\ 0 & \text{otherwise} \end{cases} \qquad (j = 1, \ldots, n) \tag{6.5}$$

$$y_i^s = -b_i + \sum_{j \in J_s} a_{ij} \qquad (i = 1, \ldots, m). \tag{6.6}$$

Definition 6.1: *The root of ordered pseudo-solution*

$$\phi = (x_{j_1} = \delta_{j_1}, \ldots, x_{j_k} = \delta_{j_k})$$

is the following vector

$$x_j^0(\phi) = \begin{cases} \delta_j \text{ if } j = j_1, \ldots, j_k \\ 0 \text{ otherwise.} \end{cases}$$

Let us consider a positive integer, t and the tth ordered pseudo-solution occurring in the algorithm ϕ_t

$$\phi_t = (\alpha_{j_1}, \alpha_{j_2}, \ldots, \alpha_{j_r}),$$

where α_{j_k} denotes one of the variable fixings $x_{j_k} = 0$ or $x_{j_k} = 1$. Now we wish to determine the set of subscripts of the free variables that may be considered for the next fixing at value 1. This set will be called the candidate set.

We need to choose a variable which will push our present trial solution, the root of ϕ_t, $\mathbf{x}^0(\phi_t)$, closer to a feasible solution in some sense. For example we want to take into account only those variables increasing the feasibility at least in one row. Then the candidate set for the ordered pseudo-solution ϕ_t may be defined as

$$N_t = \{j \mid 0 < j \leqq n;\ j \neq j_1, \ldots, j_r;\ \exists i \in \{1, \ldots, m\};\ a_{ij} > 0,\ y_i^t < 0;\ c_j < z^* - c^T x^t\}, \tag{6.7}$$

where $\mathbf{x}^t = \mathbf{x}^0(\phi_t)$ and $\mathbf{y}^t = \mathbf{A}\mathbf{x}^t - \mathbf{b}$, is the trial solution corresponding to the ordered pseudo-solution ϕ_t.

We have to choose one of the variables $x_j, j \in N_t$. For this purpose let us introduce the following evaluation for the candidate variables

$$v_j^t = \sum_i (y_i^t + a_{ij})^- \quad (j \in N_t) \tag{6.8}$$

where the minus sign in the exponent means the negative part $d^- = \min(d, 0)$. Obviously if the solution obtained by the new fixing $x_j = 1$

$$x^0(\phi_t, x_j = 1) \tag{6.9}$$

is feasible, then $v_j^t = 0$. Otherwise v_j^t is a measure of infeasibility for vector (6.9). Therefore the variable $x_{j_{t+1}^*}$ with the smallest infeasibility is chosen

$$\max_{j \in N_t} v_j^t = v_{j_{t+1}^*}^t. \tag{6.10}$$

It should be noted however, that the infeasibility may temporarily become worse. For example

$$\mathbf{y}^t = \begin{pmatrix} -2 \\ -3 \end{pmatrix} \quad \mathbf{a}_1 = \begin{pmatrix} 10 \\ -5 \end{pmatrix} \quad \mathbf{a}_2 = \begin{pmatrix} -5 \\ 10 \end{pmatrix}.$$

The present infeasibility is $v = -5$. The infeasibility measures for the two variables

$$v_1^t = (-2 + 10)^- + (-3 - 5)^- = -8$$

$$v_2^t = (-2 - 5)^- + (-3 + 10)^- = -7$$

and nevertheless by introducing both variables x_1 and x_2 a feasible solution is obtained

$$\mathbf{y}^{t+2} = \begin{pmatrix} -2 \\ -3 \end{pmatrix} + \begin{pmatrix} 10 \\ -5 \end{pmatrix} + \begin{pmatrix} -5 \\ 10 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

Now let us suppose that a new feasible solution $(\mathbf{x}^t, \mathbf{y}^t)$ has been obtained, that is, $\mathbf{y}^t \geqq \mathbf{0}$. Then an adjustment is performed on the objective function constraint: $z^* = \mathbf{c}^T\mathbf{x}^t$ and $\mathbf{x}^* = \mathbf{x}^t$. The candidate set N_k $(k < t)$ may also be reduced by the set

$$D_k^t = \{j \mid j \notin J_k, \quad c_j \geqq \mathbf{c}^T\mathbf{x}^t - \mathbf{c}^T\mathbf{x}^k\} \tag{6.11}$$

where J_k denotes the set of subscripts for the variables fixed in ordered pseudo-solution ϕ_k.

Let us denote the objective function value corresponding to the root of ordered pseudo-solution ϕ_k, by z_k

$$z_k = \mathbf{c}^T\mathbf{x}^0(\phi_k),$$

furthermore the objective function value of the best solution obtained so far (until the ordered pseudo-solution ϕ_k is formed) is denoted by $z^{(k)}$, that is,

$$z^{(k)} = \min \{\mathbf{c}^T\mathbf{x}^0(\phi_r) \mid r \leqq k, \quad \mathbf{A}\mathbf{x}^0(\phi_r) - \mathbf{b} \geqq \mathbf{0}\}.$$

If

$$\mathbf{A}\mathbf{x}^0(\phi_r) - \mathbf{b} \ngeqq \mathbf{0} \qquad \text{for all} \qquad r \leqq k,$$

then let $z^{(k)} = \infty$.

6.3 Description of the algorithm

The algorithm for solving (6.1)–(6.4) may be summarized in the following steps:

Step 1: If the vector $\mathbf{x}^0 = \mathbf{0}$ is a feasible solution ($\mathbf{y}^0 = \mathbf{b} \geqq \mathbf{0}$) then it is also optimal. Then $\mathbf{x}^* = \mathbf{0}$, $z^* = 0$ and go to *Step* 7. Otherwise let

$$\phi_0 = (\emptyset), \quad t = 0, \quad z^* = -1 - \sum_{j=1}^{n} c_j.$$

Go to *Step* 3.

Step 2: If $\mathbf{y}^t \geqq \mathbf{0}$, then let $\mathbf{x}^* = \mathbf{x}^t$, $z^* = \mathbf{c}^T\mathbf{x}^t$, $N_t = \emptyset$, and the objective function constraint $-\mathbf{c}^T\mathbf{x} \geqq -z^* + \varepsilon$ is used.

Form the sets D_k^t according to expression (6.11) for all $k < t$ and $\{\phi_t\} \subset \{\phi_k\}$.† Fix and tie (underline) these variables at zero and add these terms to the corresponding pseudo-solutions. (The term $\underline{x_j = 0}\ j \in D_k^t$ is added just after the last term of the pseudo-solution ϕ_k). If a variable is fixed at the same value several times, then only the first term is retained. If a variable occurs twice with opposite fixing then the second fixing and all the terms following it are omitted from the present pseudo-solution. In all cases, continue at the next step.

Step 3: Form the candidate set N_t for the present solution ϕ_t according to expression (6.7). If $N_t = \emptyset$, then go to *Step* 6, otherwise to the next one.

Step 4: Check the conditions

$$\sum_{j \in N_t} a_{ij}^+ \geqq -y_i^t \qquad \text{for all} \qquad i : y_i^t < 0, \tag{6.12}$$

where the plus sign denotes the positive part $d^+ = \max\{d, 0\}$.

(a) If at least one of the conditions (6.12) is not satisfied, then let $N_t = \emptyset$ and go to *Step* 6.

† This means that all variable fixing terms of ϕ_k may also be found in ϕ_t.

(b) If all conditions (6.12) are satisfied as strict inequalities, then calculate the evaluation v_j^t, $j \in N_t$ according to expression (6.8) and choose the best one, j_{t+1}^*

$$v_{j_{t+1}^*}^t = \max_{j \in N_t} v_j^t .$$

Furthermore let

$$\phi_{t+1} = (\phi_t, j_{t+1}^*)$$

$$\mathbf{x}^{t+1} = \mathbf{x}^0(\phi_{t+1}); \quad \mathbf{y}^{t+1} = \mathbf{A}\mathbf{x}^{t+1} - \mathbf{b}.$$

Increase t by one and go to *Step* 2.

(c) Finally if all conditions (6.12) are satisfied and

$$M^t = \{i \mid y_i^t < 0, \quad \sum_{j \in N_t} a_{ij}^+ = -y_i^t\} \neq \emptyset \tag{6.13}$$

then continue with the next step.

Step 5: Check the inequality

$$\sum_{j \in F_t} c_j < z^{(t)} - z_t \tag{6.14}$$

where

$$F_t = \{j \mid j \in N_t;\ \exists\, i \in M^t : a_{ij} > 0\} = \{j_1^t, j_2^t, \ldots, j_{r_t}^t\} .$$

(a) If inequality (6.14) is satisfied, then let

$$\phi_{t+1} = (\phi_t, j_1^t, j_2^t, \ldots, j_{r_t}^t),$$

where the subscript j stands for the variable fixing $x_j = 1$ (all the newly fixed variables are also tied), and

$$\mathbf{x}^{t+1} = \mathbf{x}^0(\phi_{t+1}), \quad \mathbf{y}^{t+1} = \mathbf{A}\mathbf{x}^{t+1} - \mathbf{b}.$$

Increase t by one and go to *Step* 2.

(b) If inequality (6.14) is not satisfied, then let $N_t = \emptyset$ and go to the next step.

Step 6: If all fixed variables are also tied (underlined) then go to Step 7. Otherwise the next pseudo-solution ϕ_{t+1} is calculated as follows. The last fixed but not tied variable is changed to its opposite value and tied. All variable fixings following it are deleted, the corresponding variables are considered as free variables. Increase t by one. Calculate the vectors $\mathbf{x}^t$ and $\mathbf{y}^t$ and the candidate set N_t. If $N_t = \emptyset$, then Step 6 is repeated, otherwise go to Step 2.

Step 7: End of the algorithm. If

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

then there is no solution. Otherwise x^* is an optimal solution of (6.1)–(6.4) with the objective function value z^*.

6.4 Justification of the algorithm

As this algorithm is formulated in the framework of the algorithm family described in Section 2.5, only two special features need justification. One of them is the use of the objective function constraint in order to obtain increasingly better feasible solutions instead of determining all feasible solutions. For this purpose let us introduce the following sets

$$S^0 = S$$

and

$$S^{k+1} = \{\mathbf{x} \mid \mathbf{x} \in S^0, \quad \mathbf{c}^T\mathbf{x} < \mathbf{c}^T\mathbf{x}^k\} \qquad (k = 0, 1, \ldots, r), \tag{6.15}$$

where the set S^{k+1} is defined whenever a feasible solution

$$\mathbf{x}^k \in S^k$$

is found. In other words, the algorithm described in Section 2.5 is used solely as a means of finding just one feasible solution for each problem

$$\min_{\mathbf{x} \in S^k} \mathbf{c}^T\mathbf{x},$$

and the solution set is changed according to definition (6.15). Obviously as the process is guaranted to result in all feasible solutions within a finite number of steps, either a feasible solution $\mathbf{x}^k \in S^k$ will be found or it will be shown that no feasible solution exists. On the other hand only a finite number of sets S^k may be defined because set S contains only a finite number of elements and the feaxible solutions already found are excluded from further consideration by definition (6.15).

The other new feature is the use of the sets N_k and D_k^t. Obviously at least one element of set N_k must be chosen in order to obtain a feasible solution. The use of sets D_k^t may easily be justified in the following way. Let us imagine that after obtaining a new feasible solution $\mathbf{x}^k$ we restart the algorithm from the very beginning. In this case the choice $x_j = 1, j \in D_k^t$ for the pseudo-solution ϕ_t is obviously prohibited and thus certain short cuts will take place (compared with the case when only $\mathbf{x}^{k-1}$ was known), otherwise the algorithm will follow the same path. The procedure described in the foregoing section differs from this approach only in the respect that when optimizing on set S^{k+1}, the steps which are *exactly* the same as when optimizing on set S^k are deleted.

6.5 A numerical example

Let us solve the following problem by the modified additive algorithm

$$\min\ 3x_1 + 5x_2 + 3x_3 + 9x_4 + 5x_5 + 8x_6 + 7x_7 + 4x_8$$

$$-x_1 + 3x_2 - 5x_3 + 7x_4 - 3x_5 + 2x_6 + 5x_7 + 6x_8 - y_1 = 1$$

$$3x_1 + 3x_2 - 10x_3 - 5x_4 + 3x_5 + 5x_6 - 2x_7 - 5x_8 - y_2 = -10$$

$$2x_1 - 2x_2 + 5x_3 - 2x_4 - 4x_5 + x_6 + 3x_7 + 5x_8 - y_3 = 10$$

$$x_j \in \{0, 1\} \qquad (j = 1, \ldots, 8)$$

$$y_i \geqq 0 \qquad (i = 1, 2, 3).$$

Now the algorithm of Section 6.3 is applied

1. $\mathbf{x}^0 = \mathbf{0}, \mathbf{y}^0 = (-1, 10, -10) \ngeq \mathbf{0}$ $\phi_0 = (\emptyset)$, $t = 0$, $z^* = -45$

3. $N_0 = \{1, 2, 3, 4, 6, 7, 8\}$

4. $3 + 7 + 2 + 5 + 6 > 1$
   $2 + 5 + 1 + 3 + 5 > 10$

4(b). $v_1^0 = -10$, $v_2^0 = -12$, $v_3^0 = -11$, $v_4^0 = -12$,
$v_6^0 = -9$, $v_7^0 = -7$, $v_8^0 = -5$,

$\max_{j \in N_0} v_j^0 = v_8^0$, $j_1^* = 8$, $\phi_1 = (8)$

$\mathbf{x}^1 = (0, 0, 0, 0, 0, 0, 0, 1)$, $\mathbf{y}^1 = (5, 5, -5)$, $t = 1$

2(b). $\mathbf{y}^1 \ngeq \mathbf{0}$

3. $N_1 = \{1, 3, 6, 7\}$

4. $2 + 5 + 1 + 3 > 5$

4(b). $v_1^1 = -3$, $v_3^1 = -5$, $v_6^1 = -4$, $v_7^1 = -2$

$\max_{j \in N_1} v_j^1 = v_7^1$, $j_2^* = 7$, $\phi_2 = (8, 7)$

$\mathbf{x}^2 = (0, 0, 0, 0, 0, 0, 1, 1)$, $y^2 = (10, 3, -2)$, $t = 2$

2(b). $\mathbf{y}^2 \ngeq \mathbf{0}$

3. $N_2 = \{1, 3, 6\}$

4. $2 + 5 + 1 > 2$

TABLE 6.1

t	ϕ_s	y				z^*	x^*								v_j^t							
		1	2	3	4		1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
0		−1	10	−10		−45									−10	−12	−11	−12		−9	−7	[−5]
1	8	5	5	−5											−3		−5			−4	[−2]	
2	8 7	10	3	−2											[0]		−7			−1		
3	8 7 1	9	6	0		−14	1	0	0	0	0	0	1	1								
4	8 7 $\bar{1}$ $\bar{3}$ $\bar{6}$	10	3	−2	2																	
5	8 $\bar{7}$	5	5	−5	9										[−1]		−5			−4		
6	8 $\bar{7}$ 1	4	8	−3	6												[ ]					
7	8 $\bar{7}$ 1 3	−1	−2	0	3																	
8	8 $\bar{7}$ $\bar{1}$	5	5	−5	9												−5			[−4]		
9	8 $\bar{7}$ $\bar{1}$ 6	7	10	−4	1																	
10	8 $\bar{7}$ $\bar{1}$ $\bar{6}$	5	5	−5	9												[ ]					
11	8 $\bar{7}$ $\bar{1}$ $\bar{6}$ 3	0	−5	0	6																	
12	8 $\bar{7}$ $\bar{1}$ $\bar{6}$ $\bar{3}$	5	5	−5	9																	
13	$\bar{8}$	−1	10	−10	13										−10	−12	−11	−12			−9	−7
14	$\bar{8}$ 7	4	8	−7	6												[ ]					
15	$\bar{8}$ 7 1 3	−2	1	0	0																	
16	$\bar{8}$ $\bar{7}$	−1	10	−10	13																	

4(b). $v_1^2 = 0 \quad v_3^2 = -7 \quad v_6^2 = -1$

$\max_{j \in N_2} v_j^2 = v_1^2,\ j_3^* = 1, \quad \phi_3 = (8, 7, 1)$

$\mathbf{x}^3 = (1, 0, 0, 0, 0, 1, 1), \quad y^3 = (9, 6, 0), \quad t = 3$

2. $\mathbf{y}^3 \geqq \mathbf{0}, \quad \mathbf{x}^* = (1, 0, 0, 0, 0, 0, 1, 1), \quad z^* = 14.$

The objective function constraint is used from now on

$$-3x_1 - 5x_2 - 3x_3 - 9x_4 - 5x_5 - 8x_6 - 7x_7 - 4x_8 - y_4 \geqq -z^* + 1\,.$$

The sets D_k^i are determined from expression (6.11)

$$D_0^3 = \emptyset, \quad D_1^3 = \emptyset, \quad D_2^3 = \{1, 2, 3, 4, 5, 6,\}\,.$$

All variables having subscripts taken from set D_k^i are fixed and tied at value 0 and placed after the variable fixing j_k. (In our case only D_2^3 is nonempty)

$$\phi_4 = (8, 7, \underline{\bar{1}}, \underline{\bar{2}}, \underline{\bar{3}}, \underline{\bar{4}}, \underline{\bar{5}}, \underline{\bar{6}}), \quad t = 4\,.$$

The last variable fixing $x_1 = 1$ (in brief, term 1) was deleted because of the foregoing term $\underline{\bar{1}}$.

3. $N_4 = \emptyset$

6(b). $\phi_5 = (8, \underline{\bar{7}}), \quad t = 5$

$\mathbf{x}^5 = (0\,,0, 0, 0, 0, 0, 0, 1) \quad \mathbf{y}^5 = (5, 5, -5, 9)$

3. $N_5 = \{1, 3, 6\}$

4(b). $v_1^5 = -3, \quad v_3^5 = -5, \quad v_6^5 = -4$

$\max_{j \in N_5} v_j^5 = v_1^5 \quad j_6^* = 1 \quad \phi_6 = (8, \underline{\bar{7}}, 1), \quad t = 6$

$\mathbf{x}_6 = (1, 0, 0, 0, 0, 0, 0, 1) \quad \mathbf{y}_6 = (4, 8, -3, 6)$

2(b). $\mathbf{y}^6 \ngeqq \mathbf{0}$

3. $N_6 = \{3\}$

$\phi_7 = \{8, \underline{\bar{7}}, 1, 3\}\,.$

The algorithm may be continued in a similar way. It is summarized in Table 6.1. The tableau speaks for itself. The last column contains the evaluating numbers v_j^i. The one which is chosen is framed. An empty frame means either a single element or the occurrence of case 4(c), when all elements must be chosen. In neither of these cases should the value v_j^i be calculated.

CHAPTER 7

BENDERS DECOMPOSITION

7.1 Introduction

The procedure described in this chapter may be applied for decomposing all problems consisting of two parts (at least one of which is linear) in each constraint, and in the objective function. Discrete functions may also appear therefore the method can also be used for mixed integer problems. As a result of the decomposition a pure linear and a pure discrete programming problem is to be solved. Except for very small problems this approach is not practical. Another algorithm is also developed, based on the same theory, which coinsists of solving a series of linear and discrete (or nonlinear) programming problems of the same type.

The method is based on the work of Benders (1962) with some modifications. The chapter is completed with a summary of the algorithm, and a numerical example solved in two different ways.

7.2 Problem formulation

Let us consider the following problem

$$\max \mathbf{c}^T\mathbf{x} + f(\mathbf{y}) \tag{7.1}$$

$$\mathbf{A}\mathbf{x} + \mathbf{F}(\mathbf{y}) \leqq \mathbf{b} \tag{7.2}$$

$$\mathbf{x} \geqq \mathbf{0} \tag{7.3}$$

$$\mathbf{y} \in S, \tag{7.4}$$

where $\mathbf{A}$ is an $m \times p$ matrix, $\mathbf{x}$ and $\mathbf{y}$ are p and q component variable vectors, respectively. Accordingly $f(\mathbf{y})$ is a scalar function and $\mathbf{F}(\mathbf{y})$ is an m component vector function of q variables; $\mathbf{c}$ and $\mathbf{b}$ are given vectors of corresponding size; S is an arbitrary (possibly discrete) subset of the q-dimensional Euclidean space.

The objective function may be reduced to a single variable by introducing an additional inequality

$$x_0 - \mathbf{c}^T\mathbf{x} - f(\mathbf{y}) \leqq \mathbf{0} \tag{7.5}$$

$$\max x_0 . \tag{7.6}$$

Throughout the present chapter, (7.2)–(7.6) will be referred to as Problem (i).

We shall need the following polyhedral cones† C and C_0 of size $(m+1)$ and m, respectively, furthermore the polyhedron P, which may sometimes be empty.

† A subset K of a Euclidean space is called a polyhedral cone if it is a polyhedral set, i.e. the intersection of a finite number of closed half spaces, and it is a cone, that is, $\mathbf{u} \in K$ implies that $\lambda\mathbf{u} \in K$ for any $\lambda \geqq 0$.

$$C = \{(u_0, \mathbf{u}) \mid \mathbf{A}^T\mathbf{u} - \mathbf{c}u_0 \geqq \mathbf{0},\ \mathbf{u} \geqq \mathbf{0},\ u_0 \geqq 0\} \tag{7.7}$$

$$C_0 = \{u \mid \mathbf{A}^T\mathbf{u} \geqq \mathbf{0},\ \mathbf{u} \geqq \mathbf{0}\} \tag{7.8}$$

$$P = \{u \mid \mathbf{A}^T\mathbf{u} \geqq \mathbf{c},\ \mathbf{u} \geqq \mathbf{0}\} \tag{7.9}$$

7.3 A reformulation of the problem

For the proof of the main theorem we shall need the following lemma.

Lemma 7.1: *Let x_0^* be an arbitrary real number and $\mathbf{y}^* \in S$ an arbitrary vector. Then there exists a solution of Problem* (i) *of form $(x_0^*, \mathbf{x}, \mathbf{y}^*)$ if and only if the inequality*

$$u_0 x_0^* + \mathbf{u}^T\mathbf{F}(\mathbf{y}^*) - u_0 f(\mathbf{y}^*) \leqq \mathbf{u}^T\mathbf{b} \tag{7.10}$$

is satisfied for any vector $(u_0, \mathbf{u}) \in C$.

In other words, the lemma states that under the above conditions the optimum of Problem (i), if $\mathbf{y} = \mathbf{y}^*$ is substituted, is not worse than x_0^*.

In the proof of the lemma we shall use the following well-known theorem:

Theorem of Farkas: *The system of inequalities*

$$\mathbf{D}\mathbf{w} \geqq \mathbf{0} \tag{7.11}$$

implies the single inequality

$$\mathbf{d}^T\mathbf{w} \geqq 0, \tag{7.12}$$

that is, *all solutions* $\mathbf{w}$ *of system* (7.11) *satisfy inequality* (7.12) *if and only if there exists a vector*

$$\mathbf{v} \geqq \mathbf{0} \tag{7.13}$$

satisfying the following system of equations

$$\mathbf{v}^T\mathbf{D} = \mathbf{d}^T. \tag{7.14}$$

The proof of the theorem may be found in Farkas (1902) and in many books on linear programming, e.g. Dantzig (1963) and Prékopa (in press).

Proof of lemma 7.1: The statement of the lemma may also be formulated, that there exists a solution to the system of inequalities

$$\mathbf{A}\mathbf{x} \leqq \mathbf{b} - \mathbf{F}(\mathbf{y}^*) \tag{7.15}$$

$$-\mathbf{c}^T\mathbf{x} \leqq -x_0^* + f(\mathbf{y}^*) \tag{7.16}$$

$$\mathbf{x} \geqq \mathbf{0} \tag{7.17}$$

if and only if inequality (7.10) is satisfied for any $(u_0, \mathbf{u}) \in C$. But the system (7.15)

may be written as a system of equations with nonnegative variables, if slack variables are introduced

$$\mathbf{Ax} + \mathbf{z} = \mathbf{b} - \mathbf{F}(\mathbf{y}^*) \tag{7.18}$$

$$-\mathbf{c}^T\mathbf{x} + z_0 = -x_0^* + f(\mathbf{y}^*) \tag{7.19}$$

$$(\mathbf{x}^T, \mathbf{z}^T, z_0) \geqq \mathbf{0}\,. \tag{7.20}$$

Applying the theorem of Farkas for the following matrices

$$\mathbf{D} = \begin{pmatrix} \mathbf{A} & -\mathbf{c} \\ \mathbf{I} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{pmatrix} \quad \mathbf{V} = \begin{pmatrix} \mathbf{x} \\ \mathbf{z} \\ z_0 \end{pmatrix} \quad \mathbf{d} = \begin{pmatrix} \mathbf{b} - \mathbf{F}(\mathbf{y}^*) \\ -x_0^* + f(\mathbf{y})^* \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} \mathbf{u} \\ u^0 \end{pmatrix}$$

the inequality system (7.11) becomes the polyhedral cone (7.7), thus the lemma is proved.

Let us denote by G the set of all points $(x_0, \mathbf{y})$ for which Problem (i) can be solved. Then Lemma 7.1 may be formulated in the following way

$$G = \bigcap_{(u_0,\mathbf{u}) \in C} \{(x_0, \mathbf{y}) \mid u_0 x_0 + \mathbf{u}^T\mathbf{F}(\mathbf{y}) - u_0 f(\mathbf{y}) \leqq \mathbf{u}^T\mathbf{b}, \quad \mathbf{y} \in S\}\,. \tag{7.21}$$

According to Minkowski's theorem (1896), the polyhedral cone C can be written as a nonnegative linear combination of a finite number of vectors. Instead of expression (7.21) it is sufficient to consider only these vectors, say $(\mathbf{u}_0^k, \mathbf{u}^k)$, $(k = 1, 2, \ldots, N)$, because if the inequalities are satisfied for these vectors then so are their nonnegative linear combinations. In other words

$$G = \bigcap_{k=1}^{N} \{(x_0, \mathbf{y}) \mid u_0^k x_0 + (\mathbf{u}^k)^T\mathbf{F}(\mathbf{y}) - u_0^k f(\mathbf{y}) \leqq (\mathbf{u}^k)^T\mathbf{b}, \quad \mathbf{y} \in S\} \tag{7.22}$$

Let us consider now the following two problems

Problem (ii)

$$\max \{x_0 \mid (x_0, \mathbf{y}) \in G\} \tag{7.23}$$

Problem (iii/$\mathbf{y}$*)*

$$\max \mathbf{c}^T\mathbf{x} \tag{7.24}$$

$$\mathbf{Ax} \leqq \mathbf{b} - \mathbf{F}(\mathbf{y}) \tag{7.25}$$

$$\mathbf{x} \geqq \mathbf{0}\,. \tag{7.26}$$

As in Problem (iii/$\mathbf{y}$), the value of $\mathbf{y}$ is given; this is a linear programming problem. The main purpose of the present chapter is to decompose Problem (i) to the above two problems. Let us solve Problem (ii) first. If it has an optimum x_0^* with the optimal solution $\mathbf{y}^*$, then solve Problem (iii/$\mathbf{y}^*$) and denote the optimal solution by $\mathbf{x}^*$. Then

$$(x_0^*, \mathbf{x}^*, \mathbf{y}^*)$$

is an optimal solution of Problem (i). This is stated in the following main theorem.

Theorem 7.1: (Partitioning).

1° *There exists* (a) *a feasible solution,* (b) *a feasible solution with unbounded objective function,* (c) *an optimal solution with* x_0^* *optimum to Problem* (i) *if and only if Problem* (ii) *has a solution of type* (a), (b), *and* (c) *respectively.*

2° *If* $(x_0^*, \mathbf{y}^*)$ *is an optimal solution for Problem* (ii), *then there exists a solution* $\mathbf{x}^*$ *for Problem* (iii/$\mathbf{y}^*$) *and*

$$\mathbf{c}^T\mathbf{x}^* = x_0^* - f(\mathbf{y}^*), \tag{7.27}$$

thus $(\mathbf{x}^*, \mathbf{y}^*)$ *is an optimal solution of Problem* (i).

3° *On the other hand, if* $(\mathbf{x}^*, \mathbf{y}^*)$ *is an optimal solution of Problem* (i), *then if the number*

$$x_0^* = \mathbf{c}^T\mathbf{x}^* + f(\mathbf{y}^*) \tag{7.28}$$

is calculated, the vector $(x_0^*, \mathbf{y}^*)$ *is an optimal solution of Problem* (ii) *and* x^* *is an optimal solution of Problem* (iii/$\mathbf{y}^*$).

Proof. 1°: This statement has already been proved in Lemma 7.1.

2° As $(x_0^*, \mathbf{y}^*) \in G$, according to Lemma 7.1, there exists an optimal solution $\mathbf{x}^*$ for Problem (iii/$\mathbf{y}^*$) for which

$$x_0^* - \mathbf{c}^T\mathbf{x}^* - f(\mathbf{y}^*) = 0.$$

The vector $(x_0^*, \mathbf{x}^*, \mathbf{y}^*)$ is an optimal solution of *Problem* (i) because of 1° (c).

3° It is an obvious consequence of Lemma 7.1.

If set G can be determined according to expression (7.22), then Problem (i) can be solved in two parts: by solving consecutively Problem (ii) and Problem (iii/$\mathbf{y}^*$), where $\mathbf{y}^*$ is the solution of Problem (ii.) Then a mixed problem is reduced to the solution of two "pure" problems. For example, instead of a mixed integer programming problem a pure discrete and a linear programming problem should be solved. The determination of set G however, i.e. to give the Minkowski decomposition for larger matrices, is a hopeless task. For this reason we shall try to determine a suitable set

$$Q \subset C \tag{7.29}$$

and to calculate the transformed set $G(Q)$. Then it will be shown, that

$$G \subset G(Q). \tag{7.30}$$

After solving Problem (ii) on set $G(Q)$ instead of solving it on the set G, the optimal solution is checked, i.e. whether $(x_0^*, \mathbf{y}^*) \in G$. If the answer is positive, Problem (ii) is solved. In the opposite case there exists a vector $(u_0, \mathbf{u}) \in C$, such that

$$u_0(x_0^* - f(\mathbf{y}^*)) + \mathbf{u}^T(\mathbf{F}(\mathbf{y}^*) - \mathbf{b}) > 0. \tag{7.31}$$

Then form the new set

$$Q_1 = Q \cup \{(u_0, \mathbf{u})\}$$

and solve again Problem (ii), now on set Q_1. As $(x_0^*, \mathbf{y}^*) \notin G(Q_1)$, another solution must be obtained. The process is continued until the optimization on the set $G(Q_k)$ results in a point of set G. This is a sketch of the procedure which will be worked out in detail.

7.4 Preliminary lemmata

For simplicity, let us suppose, that the following conditions hold:

(i) *Set S is compact, i.e. bounded and closed.*

(ii) *Functions $f(\mathbf{y})$ and $\mathbf{F}(\mathbf{y})$ are continuous on set S.*

If set S consists of a finite number of elements, then the above conditions obviously hold. For the justification of the procedure outlined in the foregoing section, we have to prove a few preliminary lemmata.

Lemma 7.2: *Let us suppose that there exists a feasible solution for Problem* (ii). *This problem has an unbounded solution if and only if $P = \emptyset$.*

Proof. According to the supposition there exists a point

$$(x_0^*, \mathbf{y}^*) \in G.$$

If polyhedron P is empty, then $u_0 = 0$ for any $(u_0, \mathbf{u}) \in C$. But then

$$(\lambda x_0^*, \mathbf{y}^*) \in G \qquad \text{for any} \qquad \lambda \geqq 0,$$

that is, Problem (ii) is unbounded.

If polyhedron P is nonempty, then there exists an element

$$(1, \mathbf{u}^*) \in C.$$

It is obvious from equation (7.21) that

$$G \subset G_1 = \{(x_0, \mathbf{y}) \mid x_0 \leqq f(\mathbf{y}) - \mathbf{u}^{*T}\mathbf{F}(\mathbf{y}) - \mathbf{u}^{*T}\mathbf{b}, \quad \mathbf{y} \in S\}.$$

As a consequence of the above conditions (A) and (B), set G_1 is bounded, then so are set G and the feasibility domain of Problem (ii).

Lemma 7.3: *Let $Q \subset C$ be an arbitrary nonempty set and define*

$$G(Q) = \bigcap_{(u_0, \mathbf{u}) \in Q} \{(x_0, \mathbf{y}) \mid u_0 x_0 + \mathbf{u}^T\mathbf{F}(\mathbf{y}) - u_0 f(\mathbf{y}) \leqq \mathbf{u}^T\mathbf{b}, \mathbf{y} \in S\}. \tag{7.32}$$

If the problem

$$\max \{x_0 \mid (x_0, \mathbf{y}) \in G(Q)\} \tag{7.33}$$

has no solution, then neither does Problem (ii). *In other words, equation* $G(Q) = \emptyset$ *implies that* $G = \emptyset$.

Proof: As $Q \subset C$, definitions (7.21) and (7.32) imply that

$$G(Q) \supset G, \tag{7.34}$$

which proves the lemma.

Lemma 7.4: *Let vector* $(x_0^*, \mathbf{y}^*)$ *be an optimal solution of problem* (7.33). *This vector is also an optimal solution of Problem* (ii) *if and only if*

$$\min \{(\mathbf{b} - \mathbf{F}(\mathbf{y}^*))^T \mathbf{u} \mid \mathbf{u} \in P\} = x_{0i}^* - f(\mathbf{y}^*). \tag{7.35}$$

Proof: As $G(Q) \supset G$, $(x_0^*, \mathbf{y}^*)$ is an optimal solution of Problem (ii) if and only if $(x_0^*, \mathbf{y}^*) \in G$.

To show the necessity of condition (7.35), let us suppose that $(x_0^*, \mathbf{y}^*)$ is an optimal solution of Problem (ii), that is, $(x_0^*, \mathbf{y}^*) \in G$

$$\mathbf{u}^T(\mathbf{b} - \mathbf{F}(\mathbf{y}^*)) \geqq u_0(x_0^* - f(\mathbf{y}^*)) \quad \text{for any} \quad (u_0, \mathbf{u}) \in C. \tag{7.36}$$

If $u_0 \neq 0$, then inequality (7.36) may be divided by u_0

$$\mathbf{u}^T(\mathbf{b} - \mathbf{F}(\mathbf{y}^*)) \geqq x_0^* - f(\mathbf{y}^*) \quad \text{for any} \quad \mathbf{u} \in P, \tag{7.37}$$

as $u_0 > 0$ and C is a cone. If $u_0 = 0$ we obtain

$$\mathbf{u}^T(\mathbf{b} - \mathbf{F}(\mathbf{y}^*)) \geqq 0 \quad \text{for any} \quad \mathbf{u} \in C_0. \tag{7.38}$$

The set of inequalities (7.37) implies, that

$$\min \{\mathbf{u}^T(\mathbf{b} - \mathbf{F}(\mathbf{y}^*)) \mid \mathbf{u} \in P\} \geqq x_0^* - f(\mathbf{y}^*). \tag{7.39}$$

On the other hand, the problem on the left hand side of equation (7.35) may be written as

$$\min (\mathbf{b} - \mathbf{F}(\mathbf{y}^*)^T \mathbf{u} \tag{7.40}$$

$$\mathbf{A}^T \mathbf{u} \geqq \mathbf{c} \tag{7.41}$$

$$\mathbf{u} \geqq \mathbf{0}. \tag{7.42}$$

The dual of (7.40)–(7.42) is the following

$$\max \mathbf{c}^T \mathbf{x} \tag{7.43}$$

$$\mathbf{A}\mathbf{x} \leqq \mathbf{b} - \mathbf{F}(\mathbf{y}^*) \tag{7.44}$$

$$\mathbf{x} \geqq \mathbf{0}. \tag{7.45}$$

The duality theorem of linear programming asserts (see for example Dantzig (1963) that if one of the dual problems has a bounded solution, then so does the

other and their optima are equal to each other. As a consequence of Lemma 7.2, (7.40)–(7.42) has a feasible solution. Inequality (7.39) shows that the problem is also bounded. Thus, there exists an optimal solution $\mathbf{x}^*$ to problem (7.43)–(7.45) and

$$(\mathbf{b} - \mathbf{F}(\mathbf{y}^*))^T\mathbf{u}^* = \mathbf{c}^T\mathbf{x}^*, \tag{7.46}$$

where $\mathbf{u}^*$ denotes an optimal solution of (7.40)–(7.42). As the conditions of Theorem 7.1 are satisfied,

$$\mathbf{c}^T\mathbf{x}^* = x_0^* - f(\mathbf{y}^*) \tag{7.47}$$

according to point 2° of the theorem. Equations (7.46) and (7.47) together show the necessity for condition (7.35).

To see the sufficiency, let us suppose that equation (7.35) is satisfied. This immediately implies inequalities (7.37). The inequalities (7.38) are also satisfied, as is shown by the following indirect reasoning. If there exists $\hat{\mathbf{u}} \in C_0$, for which

$$\hat{\mathbf{u}}^T(\mathbf{b} - \mathbf{F}(\mathbf{y}^*)) < 0, \tag{7.48}$$

then $\lambda\hat{\mathbf{u}} \in C_0$ for any $\lambda \geqq 0$, as C_0 is a cone. But this is impossible, because then for an arbitrary vector $\mathbf{u} \in P$

$$\mathbf{u} + \lambda\hat{\mathbf{u}} \in P$$

provides an unbounded solution when $\lambda \to \infty$, which is in contradiction to equation (7.35).

If we now reverse the reasoning of the sufficiency part, inequalities (7.37) and (7.38) imply inequalities (7.36), which means, that

$$(x_1^*, \mathbf{y}^*) \in G$$

But then $(x_0^*, \mathbf{y}^*)$ is an optimal solution of Problem (ii), which proves the lemma.

A vector $\mathbf{u} \in K$ is called extremal point of polyhedron K, if there does not exist a pair of vectors $\mathbf{u}^1, \mathbf{u}^2 \in K$, $\mathbf{u}^1 \neq \mathbf{u}$ and $\mathbf{u}^2 \neq \mathbf{u}$, for which

$$\mathbf{u} = \lambda\mathbf{u}^1 + (1 - \lambda)\mathbf{u}^2 \qquad 0 < \lambda < 1\,.$$

Similarly, vector $\mathbf{v}$ is called the extremal direction of polyhedron K, if there exists a vector $\mathbf{u}$ for which

$$\mathbf{u} + \mu\mathbf{v} \in K \quad \text{for any} \quad \mu \geqq 0,$$

but there does not exist a pair of feasible directions $\mathbf{v}^1 \neq \alpha\mathbf{v}$ and $\mathbf{v}^2 \neq \alpha\mathbf{v}$ ($\alpha \geqq 0$ arbitrary real number), for which

$$\mathbf{u} + \mu\mathbf{v}^1 \in K \text{ and } \mathbf{u} + \mu\mathbf{v}^2 \in K \quad \text{for any} \quad \mu \geqq 0$$

and furthermore

$$\mathbf{v} = \lambda\mathbf{v}^1 + (1 - \lambda)\mathbf{v}^2, \quad \text{where} \quad 0 < \lambda < 1.$$

We can now formulate our next statement.

Lemma 7.5: 1°. *The vector* $(1, \mathbf{u})$ *is an extremal direction of cone* C, *if and only if vector* $\mathbf{u}$ *is an extremal point of polyhedron* P.

2° *The vector* $(0, \mathbf{v})$ *is an extremal direction of cone* C, *if and only if vector* $\mathbf{v}$ *is an extremal direction of polyhedron* P.

Proof: It follows from definitions (7.7) and (7.9) of sets C and P that if $\mathbf{u} \in P$ then $(\mu, \mu\mathbf{u}) \in C$ for any $\mu \geqq 0$. If there exist vectors $(v_0^1, \mathbf{v}^1) \in C$ and $(v_0^2, \mathbf{v}^2) \in C$ differing from vector $(1, \mathbf{u})$ not only in a constant multiplier, such that

$$(1, \mathbf{u}) = \lambda(v_0^1, \mathbf{v}^1) + (1 - \lambda)(v_0^2, \mathbf{v}^2) \qquad (0 < \lambda < 1),$$

then $v_0^1 = v_0^2 = 0$ is impossible.

If both $v_0^1 > 0$ and $v_0^2 > 0$, then using the notation

$$\mathbf{u}^1 = \frac{1}{v_0^1}\mathbf{v}^1 \in P \quad \text{and} \quad \mathbf{u}^2 = \frac{1}{v_0^2}\mathbf{v}^2 \in P,$$

$$\mathbf{u} = \lambda v_0^1 \mathbf{u}^1 + (1 - \lambda)v_0^2\mathbf{u}^2 \quad (\lambda v_0^1 + (1 - \lambda)v_0^2 = 1),$$

which shows that u is not an extremal point of the polyhedron, because $\mathbf{u}^1 \neq \mathbf{u}$, $\mathbf{u}^2 \neq \mathbf{u}$.

On the other hand, if one of the numbers v_0^1 and v_0^2, say $v_0^2 = 0$, then

$$(1, \mathbf{u}) = \lambda(v_0^1, \mathbf{v}^1) + (1 - \lambda)(0, \mathbf{v}^2) = \frac{\lambda}{\lambda + 2} \cdot \left(\frac{2 + \lambda}{2} v_0^1, \frac{2 + \lambda}{2} \mathbf{v}^1 \right) +$$

$$+ \left(1 - \frac{\lambda}{\lambda + 2}\right) \left(\frac{\lambda(2 + \lambda)}{4} v_0^1, \frac{(1 - \lambda)(2 + \lambda)}{2} \mathbf{v}^2 + \frac{\lambda(2 + \lambda)}{4} \mathbf{v}^1 \right).$$

As

$$0 < \frac{\lambda}{\lambda + 2} < 1 \quad \text{and} \quad \frac{2 + \lambda}{2} v_0^1 > 0, \quad \frac{\lambda(2 + \lambda)}{4} v_0^1 > 0,$$

furthermore the two new vectors are also in cone C, this situation is reduced to the case when both first components are strictly positive, i.e. it was shown that if $(1, \mathbf{u})$ is not an extremal direction of cone C, then $\mathbf{u}$ is not an extremal point of polyhedron P. The opposite statement is obvious because if

$$\mathbf{u} = \lambda\mathbf{u}^1 + (1 - \lambda)\mathbf{u}^2, \quad \mathbf{u}^1, \mathbf{u}^2 \neq \mathbf{u}, \quad \mathbf{u}^1, \mathbf{u}^2 \in P,$$

then

$$(1, \mathbf{u}) = \lambda(1, \mathbf{u}^1) + (1 - \lambda)(1, \mathbf{u}^2) \quad \text{and} \quad (1, \mathbf{u}^1), (1, \mathbf{u}^2) \in C,$$

differing from $(1, \mathbf{u})$ not only in a constant multiplier.

2° Let vector $\mathbf{v}$ be an extremal direction of polyhedron P, and

$$\mathbf{u} + \mu\mathbf{v} \in P \qquad \text{for any } \mu \geqq 0.$$

Then

$$\mathbf{A}^T\mathbf{u} + \mathbf{A}^T\mathbf{v} > \mathbf{c} \qquad \text{for any } \mu \geqq 0,$$

hence

$$\mathbf{A}^T\mathbf{v} \geqq \mathbf{0},$$

which implies that $(0, \mathbf{v}) \in C$. If this was not an extremal direction of cone C, then there would exist vectors $(v_0^1, \mathbf{v}^1) \in C$ and $(v_0^2, \mathbf{v}) \in C$ differing from vector $(0, \mathbf{v})$, not only in a constant multiplier, such that

$$(0, \mathbf{v}) = \lambda(v_0^1, \mathbf{v}^1) + (1 - \lambda)(v_0^2, \mathbf{v}^2) \qquad (0 < \lambda < 1).$$

As in the definition of cone C, $(u_0, \mathbf{u}) \leqq \mathbf{0}$, thus $v_0^1 = v_0^2 = 0$. Then

$$\mathbf{A}^T\mathbf{v}^1 \geqq \mathbf{0} \quad \text{and} \quad \mathbf{A}^T\mathbf{v}^2 \geqq \mathbf{0},$$

that is, $\mathbf{v}^1$ and $\mathbf{v}^2$ are feasible directions of polyhedron P differing from v not only in a constant multiplier, and

$$\mathbf{v} = \lambda\mathbf{v}^1 + (1 - \lambda)\mathbf{v}^2,$$

which is a contradiction.

The opposite statement is obvious as in part 1°, using the vector $(0, \mathbf{u})$ instead of $(1, \mathbf{u})$, and this completes the proof of the lemma.

7.5 An algorithm for mixed integer problems

Let S be the set of all lattice points of a bounded domain in the q-dimensional Euclidean space.

Now Problem (i) is a mixed integer problem which is reduced to a series of linear programming and a series of pure discrete programming problems, solved alternately. The algorithm can be summarized in the following steps.

Step 1: Let $i = 0$ and choose an arbitrary set $Q_0 \subset C$. (Possibly $Q_0 = \emptyset$).

Step 2: Solve problem (7.33) for the set $G(Q_i)$, defined by equations (7.22).

(a) If $G(Q_i) = \emptyset$, then Problem (i) has no solution.

(b) If $G(Q_i) \neq \emptyset$, then an optimal solution of problem (7.33) is denoted by $(x_0^i, \mathbf{y}^i)$. (If $Q_0 = \emptyset$, $x_0^0 = +\infty$ and $\mathbf{y}^0$ is an arbitrary element of set S.)

Step 3: Calculate the quantity

$$z_i = \min \{(\mathbf{b} - \mathbf{F}(\mathbf{y}^i))^T\mathbf{u} \mid \mathbf{u} \in P\}, \tag{7.49}$$

and an optimal solution of this problem is denoted by $\mathbf{u}^i$.

(a) If $P = \emptyset$, then Problem (i) has no bounded solution. (It has either no solution at all or it has an unbounded solution.)

(b) If

$$z_i \geqq x_0^i - f(\mathbf{y}^i),$$

then solve Problem (iii/$\mathbf{y}^i$) and denoting its optimal solution by x^i, $(x_0^i, \mathbf{x}_i, \mathbf{y}_i)$ is an optimal solution of Problem (i).

(c) If

$$-\infty < z_i < x_0^i - f(\mathbf{y}^i),$$

then let

$$Q_{i+1} = Q_i \cup (1, \mathbf{u}^i) \tag{7.50}$$

and increase i by 1. The algorithm is continued at

Step 2:

(d) If

$$z_i = -\infty \qquad (P \neq \emptyset),$$

then there exists a pair of vectors $(\mathbf{u}^i, \mathbf{v}^i)$, such that

$$\mathbf{u}^i + \lambda \mathbf{v}^i \in P \qquad \text{for any} \quad \lambda \geqq 0$$

and

$$\lim_{\lambda \to +\infty} (\mathbf{b} - \mathbf{F}(\mathbf{y}^i))^T (\mathbf{u}^i + \lambda \mathbf{v}^i) = -\infty.$$

Such a pair of vectors is provided by the simplex method of linear programming. Then let

$$Q_{i+1} = Q_i \cup (0, \mathbf{v}^i) \cup (1, \mathbf{u}^i) \tag{7.51}$$

and increase i by 1. The algorithm is continued at *Step* 2.

Theorem 7.2: (Convergence)

1° *The above algorithm comes to an end in a finite number of steps.*

2° *If the algorithm is finished at Step 2a and 3a, then Problem (i) has no solution and no bounded solution, respectively.*

3° *If the algorithm is finished at Step 3b, then the optimal solution of Problem (i) has been obtained.*

Proof. 2° In Case 2a, if $G(Q_i) = \emptyset$, then because of the relation

$$Q_i \subset C \tag{7.52}$$

the following relation also holds

$$G(Q_i) \supset G(C) = G,$$

that is, $G = \emptyset$. Problem (ii) has no solution and thus, according to 1°(a) Theorem 7.1, neither has Problem (i). Relation (7.52) holds true because $Q_0 \subset C$ and this property is hereditary from Q_i to Q_{i+1} as can be shown with the help of Lemma 7.5.

In Case 3a consider the dual of (7.49) for an arbitrary vector $\mathbf{y} \in S$

$$\max \mathbf{c}^T\mathbf{x} \tag{7.53}$$

$$\mathbf{Ax} \leqq \mathbf{b} - \mathbf{F(y)} \tag{7.54}$$

$$\mathbf{x} \geqq \mathbf{0}. \tag{7.55}$$

Problem (7.49) has no feasible solution for any $\mathbf{y} \in S$, because $P = \emptyset$. Thus (7.53)–(7.55) has either no solution or it has unbounded solution for each $\mathbf{y} \in S$, according to the duality theorem of linear programming. The only difference between (7.53)–7.55) and Problem (i) is the function $f(\mathbf{y})$ which is a constant for a given $\mathbf{y} \in S$; thus Problem (i) either has no solution or it has unbounded one.

3° In Case 3b the condition of Lemma 7.4 is satisfied, therefore the present $(x_0^i, \mathbf{y}^i)$ is an optimal solution of Problem (ii). Thus, according to Theorem 7.1, vector $\mathbf{x}^i$, the optimal solution of Problem (iii/y^i), can be completed to an optimal solution of Problem (i)

$$(x_0, \mathbf{x}^i, \mathbf{y}^i).$$

1° C is a convex polyhedral cone therefore it can be obtained as nonnegative linear combinations of a finite number of vectors. If the number of these vectors is minimal, then these are the extremal directions of cone C. Let us denote the set of extremal directions by U

$$U = \{(u_0^1, \mathbf{u}^1), (u_0^2, \mathbf{u}^2), \ldots, (\mathbf{u}_0^N, \mathbf{u}^N)\}.$$

On comparing equations (7.21) and (7.22), we obtain that

$$G = G(C) = G(U).$$

If Cases 2a and 3a do not occur throughout the algorithm then according to Lemma 7.5 at least one new extremal direction is added to set Q_i at each step 3c or 3d. In Case 3c

$$(1, \mathbf{u}^i) \notin Q_i$$

because the inequality

$$-\infty < z_i < x_0^i - f(\mathbf{y}^i)$$

can be written as

$$(\mathbf{b} - \mathbf{F}(\mathbf{y}_0^i))^T\mathbf{u}_0^i < \mathbf{x}_0^i - f(\mathbf{y}^i).$$

In Case 3d the $-\infty$ limit implies, that

$$(\mathbf{b} - \mathbf{F}(\mathbf{y}^i))^T\mathbf{v}^i < 0,$$

that is,

$$(0, \mathbf{v}^i) \in Q_i.$$

In Case 3d the vector $(1, \mathbf{u}^i)$ may or may not belong to set Q_i, but the number of extremal directions is guaranteed to be increased at least by 1. As the

number of extremal directions is finite, after a finite number of iterations we obtain that

$$C \supset Q_i \supset U$$

Then

$$G \subset G(Q_i) \subset G(U) = G,$$

that is,

$$G(Q_i) = G.$$

In this case the algorithm is finished in a single step. Thus the proof of the theorem is complete.

Notes: In general, many fewer than all extremal directions of cone C should be determined because usually good directions are determined by solving (7.49).

The starting set Q_0 depends on the particular problem to be solved. In the worst case $Q_0 = \emptyset$ may be used, but a good set Q_0 accelerates the algorithm to a considerable extent. For this purpose, it is useful to know several good feasible solutions to the problem, which may be determined by a heuristic method.

7.6 A numerical example

Let us consider the following mixed integer programming problem

$$\begin{aligned}
\max\ & 2x_1 - 3x_2 + 5y_1 + 8y_2 + 2y_3 + 5y_4 - 3y_5 + 8y_6 \\
& 5x_1 + 2x_2 - 10y_1 + 10y_2 + 3y_3 + 12y_4 - 5y_5 + 18y_6 \leqq 30 \\
& -5x_2 + 8y_1 + 17y_2 + 6y_3 + 8y_4 + y_5 + 9y_6 \leqq 25 \\
& 8x_1 + 4y_1 + 9y_2 + 5y_3 + 2y_4 - 4y_5 + 6y_6 \leqq 20 \\
& x_j \geqq 0 \qquad (j = 1, 2) \\
& y_j \in \{0, 1\} \qquad (j = 1, \ldots, 6).
\end{aligned}$$

This problem will be solved first by the complete Minkowski decomposition and then by the iterative procedure described in Section 7.4

According to the notation of Problem (i) let

$$\mathbf{D} = (\mathbf{c} - \mathbf{A}^T).$$

To obtain set G, defined by expression (7.22), we have to determine the complete Minkowski decomposition of cone C. The method may be found, for example in Prékopa (in press). Minkowski's procedure is nothing but the explicit determination of all solutions of the inequality system

$$\mathbf{D}\begin{pmatrix} u_0 \\ \mathbf{u} \end{pmatrix} \leqq \mathbf{0} \qquad \begin{pmatrix} u_0 \\ \mathbf{u} \end{pmatrix} \geqq \mathbf{0},$$

where

$$\mathbf{D} = \begin{pmatrix} 2 & -5 & 0 & -8 \\ -3 & -2 & 5 & 0 \end{pmatrix}.$$

The transformation matrix of the first row is

$$\mathbf{H}\,(2, \quad -5, \quad 0, \quad -8) = \begin{bmatrix} 0 & 0 & 0 & 5 & 8 \\ 1 & 0 & 0 & 2 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 2 \end{bmatrix}.$$

The unit vectors correspond to the nonpositive elements. As the second, third and fourth elements are nonpositive, the unit vectors $\mathbf{e}_2$, $\mathbf{e}_3$ and $\mathbf{e}_4$ are used. The rest of the columns correspond to the positive–negative pairs. (They are used with positive sign and in reversed order.)

All nonnegative solutions of the inequality

$$2u_0 - 5u_1 + 0u_2 - 8u_3 \leqq 0$$

may be expressed with the help of the matrix

$$\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 5 & 8 \\ 1 & 0 & 0 & 2 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix} \quad v_j \geqq 0 \quad (j = 1, \ldots, 5).$$

This expression may be substituted in the second inequality of the system $\mathbf{Dz} \leqq \mathbf{0}$:

$$(-3, -2, 5, 0) \begin{bmatrix} 0 & 0 & 0 & 5 & 8 \\ 1 & 0 & 0 & 2 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix} = -2v_1 + 5v_2 + 0v_3 + 19v_4 - 24v_5 \leqq 0$$

$$v_j \geqq 0 \quad (j = 1, \ldots, 5).$$

All nonnegative solutions of this new inequality may be determined similarly

$$\mathbf{H}(-2, \ 5, \ 0, \ -19, \ -24) = \begin{bmatrix} 1 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 19 & 24 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 5 \end{bmatrix}.$$

Thus all nonnegative solutions of this new inequality may be written as

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 19 & 24 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \end{bmatrix} \qquad \begin{matrix} w_j \geq 0 \\ (j = 1, \ldots, 7). \end{matrix}$$

Substituting this expression in that of vector $(u_0, \mathbf{u})$

$$\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 5 & 8 \\ 1 & 0 & 0 & 2 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 19 & 24 \\ 0 & 0 & 1 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 5 \end{bmatrix} \mathbf{w} = \begin{bmatrix} 0 & 0 & 5 & 8 & 0 & 25 & 40 \\ 1 & 0 & 2 & 0 & 5 & 10 & 0 \\ 0 & 0 & 0 & 0 & 2 & 19 & 24 \\ 0 & 1 & 0 & 2 & 0 & 0 & 10 \end{bmatrix} \mathbf{w} \qquad \begin{matrix} w_j \geqq 0 \\ (j=1,\ldots,7). \end{matrix}$$

Thus we have obtained all nonnegative solutions of the inequality system, i.e. the explicit form of set G. At the same time the extremal directions $(u_0^k, \mathbf{u}^k)$ are available. They are the coefficient vectors of parameters $w_1, \ldots, w_7$. In other words, all points of cone C may be obtained as nonnegative linear combinations of column vectors of the above coefficient matrix.

Now the inequality system (7.22) determining set G may be written. In our case

$$f(\mathbf{y}) = (5 \quad 8 \quad 2 \quad 5 \quad -3 \quad 8)\ \mathbf{y}$$

$$\mathbf{F}(\mathbf{y}) = \begin{pmatrix} 10 & 10 & 3 & 12 & -5 & 18 \\ 8 & 17 & 6 & 8 & 1 & 9 \\ 4 & 9 & 5 & 2 & -4 & 6 \end{pmatrix} \mathbf{y}$$

$$\mathbf{b}^T = (30, 25, 20)\,.$$

Thus, set G is nothing other than the set of all points $(x_0, \mathbf{y})$ satisfying the following system of inequalities

$$\begin{aligned}
10y_1 + 10y_2 + 3y_3 + 12y_4 - 5y_5 + 18y_6 &\leqq 30 \\
4y_1 + 9y_2 + 5y_3 + 2y_4 - 4y_5 + 6y_6 &\leqq 20 \\
5x_0 - 5y_1 - 20y_2 - 4y_3 - y_4 + 5y_5 - 4y_6 &\leqq 60 \\
8x_0 - 32y_1 - 46y_2 - 6y_3 - 36y_4 + 16y_5 - 52y_6 &\leqq 40 \\
66y_1 + 84y_2 + 27y_3 + 76y_4 - 23y_5 + 108y_6 &\leqq 200 \\
25x_0 + 127y_1 + 223y_2 + 94y_3 + 147y_4 + 44y_5 + 151y_6 &\leqq 775 \\
40x_0 + 32y_1 + 178y_2 + 114y_3 + 12y_4 + 104y_5 - 44y_6 &\leqq 800
\end{aligned}$$

where

$$y_i \in \{0, 1\} \qquad (j = 1, \ldots, 6)$$

x_0^1 is an arbitrary real number.

This is Problem (ii) and it can be solved by any method constructed for pure integer problems. The only difficulty is to handle the single noninteger variable x_0. This difficulty will easily be overcome. If $G = \emptyset$, then Problem (ii) has no solution and according to Theorem 7.1, neither has Problem (i.) To determine whether set G is empty, it is sufficient to find a vector $\mathbf{y}^1$ satisfying all the above inequalities that do not contain variable x_0. (In our example $\mathbf{y}^1 = \mathbf{0}$ is suitable.) All other inequalities may be satisfied by choosing a sufficient value for variable x_0. It should be noted that no coefficient of variable $x_0(\mathbf{u}_0^k)$ can be negative.

Let us suppose, that $G \neq \emptyset$, and a vector $\mathbf{y}^1$ has been determined with the required property. Then there exists

$$x_0^1 = \max \{x_0 \mid (x_0, \mathbf{y}^1) \in G\}.$$

It can easily be seen that if

$$x_0' = \text{mas } \{x_0 \mid (x_0, \mathbf{y}') \in G\}$$

$$x_0'' = \max \{x_0 \mid (x_0, \mathbf{y}'') \in G\},$$

then

$$x_0' - x_0'' = K\delta,$$

where K is an integer and

$$\delta = \min_{u_0^k > 0} \left\{ \frac{1}{u_0^k} \right\},$$

where u_0^k is the coefficient of the x_0 in the kth inequality determining set G.

This is a consequence of the fact that $y_j = 0$ or 1; in addition, all coefficients are integer. (Note that all rows may be divided by the greatest common divisor of its coefficients. Then δ is changed accordingly.) Therefore, it is sufficient to change x_0 by δ

$$x_0^k = x_0^1 + (k - 1)\delta,$$

and to determine an element $(x_0^k, \mathbf{y}) \in G$.

If there exists an element

$$(x_0^k, \mathbf{y}^k) \in G,$$

but there is no vector

$$(x_0^{k+1}, \mathbf{y}) \in G,$$

then

$$\begin{pmatrix} x_0^* \\ \mathbf{y}^* \end{pmatrix} = \begin{pmatrix} x_0^k \\ \mathbf{y}^k \end{pmatrix}$$

is an optimal solution of Problem (ii), and Problem (iii/$\mathbf{y}^*$) provides the corresponding optimal vector $\mathbf{x}^*$. It is by no means necessary, however, to use such small

steps as δ. After obtaining the vector $(x_0^1, \mathbf{y}^1)$ we may choose a large value for x_0^2. In general, if we find a point $(x_0^k, \mathbf{y}^k) \in G$, then we may choose an arbitrary value

$$x_0^{k+1} \geqq x_0^k + \delta .$$

In the opposite case, when no point $(x_0^k, \mathbf{y}) \in G$ exists, then

$$x_0^{k+1} = \frac{x_0^k + x_0^l}{2},$$

where x_0^l is the last value for which $(x_0^l, \mathbf{y}^l) \in G$ was found. The procedure is continued until

$$x_0^k - x_0^l < \delta$$

and there exists $(x_0^l, \mathbf{y}^l) \in G$, but there is no point $(x_0^k, \mathbf{y}) \in G$. Then

$$\begin{pmatrix} x_k^* \\ \mathbf{y}^* \end{pmatrix} = \begin{pmatrix} x_0^l \\ \mathbf{y}^l \end{pmatrix}.$$

The determination of vector $\mathbf{x}^*$ takes place as before. In the procedure, a feasible solution of a linear discrete programming problem is to be determined, or it is to be shown that no solution exists. For this purpose any procedure suitable for pure discrete problems can be applied.

Without going into detail, in our problem we obtain that

$$(x_0^*, \mathbf{y}^*) = (16.04,\ 0,\ 1,\ 0,\ 0,\ 0,\ 1)^T.$$

Then Problem (iii/y*) has the following form

$$\begin{aligned} \max\ & 2x_1 - 3x_2 \\ & 5x_1 + 2x_2 \leqq 2 \\ & -5x_2 \leqq -1 \\ & 8x_1 \leqq 5 \\ & x_1 \geqq 0, \quad x_2 \geqq 0. \end{aligned}$$

Using the simplex method of linear programming we obtain that

$$\mathbf{x}^* = (0.32,\ 0.2).$$

As a check, let us calculate the quantity

$$\mathbf{c}^T\mathbf{x}^* + f(\mathbf{y}^*),$$

which really gives the value $x_0^* = 16.04$.

Now let us solve our problem by using the iterative procedure described in Section 7.4. In such a way, large problems may also be solved because the Minkowski decomposition of set C is not required.

The steps of the algorithm are the following.

Step 1: $\qquad i = 1, \quad Q_0 = \emptyset$

Step 2(b): $\qquad x_0^0 = +\infty, \quad \mathbf{y}^0 = (0, 0, 0, 1, 0, 1)\,.$

The point $\mathbf{y}^0$ has been chosen by the criterion

$$\max_{c_j > 0} \frac{c_j}{\sum_i a_{ij}}\,.$$

The first two variables have been chosen according to this evaluation. In this case, the problem obviously has a solution using, for example, $\mathbf{x} = \mathbf{0}$. Other starting points could also have been used.

Step 3:

$$\begin{array}{lllll} \min & & 8u_2 + 12u_3 & & \\ & 5u_1 & + \; 8u_3 \geqq & 2 & \\ & 2u_1 - 5u_2 & \geqq & -3 & \Big\} P \\ & u_1, \quad u_2, & u_3 \geqq & 0 & \end{array}$$

$$\text{Solution } \mathbf{u}^0 = \left(\frac{2}{5}, 0, 0\right) \qquad z_0 = 0.$$

Step 3(c): $\qquad Q_1 = \{(1, \mathbf{u}^0)\} = \{(5, 2, 0, 0)\}.$

As only the direction is important, all elements of the vector are multiplied by the least common multiple of the denominators to obtain integer components.

$$i = 1$$

Step 2:

$$5x_0 - 5y_1 - 20y_2 - 4y_3 - y_4 + 5y_5 - 4y_6 \leqq 60$$

$$y_j \in \{0, 1\} \qquad (j = 1, \ldots, 6)$$

$$\max x_0$$

$$\mathbf{y}^1 = (1, 1, 1, 1, 0, 1) \qquad x_0^1 = 18.8 \quad f(\mathbf{y}^1) = 28\,.$$

Step 3: $\qquad z_1 = \min \{-23u_1 - 23u_2 - 6u_3 \mid \mathbf{u} \in P\} = -\infty$

$$\left(\frac{2}{5}, 0, 0\right) + \lambda(1, 0, 0) \in P$$

$$\mathbf{u}^1 = \left(\frac{2}{5}, 0, 0\right), \quad \mathbf{v}^1 = (1, 0, 0)\,.$$

Step 3(d): $Q_2 = Q_1 \cup (0, v^1) = Q_1 \cup (0, 1, 0, 0)$,

as $(1, \mathbf{u}^1) \in Q_1$; $\quad i = 2$

Step 2:

$$5x_0 - 5y_1 - 20y_2 - 4y_3 - y_4 + 5y_5 - 4y_6 \leqq 60$$

$$10y_1 + 10y_2 + 3y_3 + 12y_4 - 5y_5 + 18y_6 \leqq 30$$

$$y_j \in \{0, 1\} \qquad (j = 1, \ldots, 6)$$

$$\max x_0$$

$$\mathbf{y}^2 = (1, 1, 1, 0, 0, 0) \qquad x_0^2 = 17.8 \qquad f(\mathbf{y}^2) = 15$$

Step 3:

$$z_2 = \min \{7u_1 - 6u_2 + 2u_3 \mid \mathbf{u} \in P\} = -3.1$$

$$\mathbf{u}^2 = \left(0, \frac{3}{5}, \frac{1}{4}\right)$$

$$z_2 < x_0^2 - f(\mathbf{y}^2)$$

Step 3(c):

$$Q_3 = Q_2 \cup (1, \mathbf{u}^2) = Q_2 \cup (20, 0, 12, 5)$$

$$i = 3$$

Step 2:

$$5x_0 - 5y_1 - 20y_2 - 4y_3 - y_4 + 5y_5 - 4y_6 \leqq 60$$

$$10y_1 + 10y_2 + 3y_3 + 12y_4 - 5y_5 + 18y_6 \leqq 30$$

$$20x_0 + 16y_1 + 89y_2 + 57y_3 + 6y_4 + 52y_5 - 22y_6 \leqq 400$$

$$y_j \in \{0. 1\} \qquad (j = 1, \ldots, 6)$$

$$\max x_0$$

$$\mathbf{y}^3 = (0, 1, 0, 0, 0, 1) \qquad x_0^3 = 16.65 \qquad f(\mathbf{y}^3) = 16$$

Step 3: $\quad Z_3 = \min \{2u_1 - u_2 + 5u_3 \mid \mathbf{u} \in P\} = 0.04$

$$\mathbf{u}^3 = \left(\frac{2}{5}, \frac{19}{25}, 0\right)$$

$$z_3 < x_0^3 - f(\mathbf{y}^3)$$

Step 3(c): $Q_4 = Q_3 \cup (25, 10, 19, 0)$

$$i = 4$$

Step 2:

$$
\begin{aligned}
5x_0 - 5y_1 - 20y_2 - 4y_3 - y_4 + 5y_5 - 4y_6 &\leqq 60\\
10y_1 + 10y_2 + 3y_3 + 12y_4 - 5y_5 + 18y_6 &\leqq 30\\
20x_0 + 16y_1 + 89y_2 + 57y_3 + 6y_4 - 52y_5 - 22y_6 &\leqq 400\\
25x_0 + 127y_1 + 223y_2 + 94y_3 + 147y_4 + 44y_5 + 151y_6 &\leqq 775\\
y_j \in \{0, 1\} \qquad (j = 1, \ldots, 6)&\\
\max x_0&
\end{aligned}
$$

$$\mathbf{y}^4 = (0, 1, 0, 0, 0, 1) \qquad x_0^4 = 16.04 \qquad f(\mathbf{y}^4) = 16$$

Step 3:

$$z_4 = x_0^4 - f(\mathbf{y}^4)$$

$$x_0^* = x_0^4 = 16.04$$

$$\mathbf{y}^* = \mathbf{y}^4 = (0, 1, 0, 0, 0, 1)$$

Step 3(b): Problem (iii/$\mathbf{y}^4$) should be solved:

$$
\begin{aligned}
\max\ 2x_1 - 3x_2&\\
5x_1 + 2x_2 &\leqq 2\\
-5x_2 &\leqq -1\\
8x_1 &\leqq 5\\
x_1 \geqq 0, \quad x_2 &\geqq 0
\end{aligned}
$$

$$x_1^* = 0.32 \qquad x_2^* = 0.2$$

Check: $x_0^* = \mathbf{c}^T\mathbf{x}^* + f(\mathbf{y}^*)$ holds.

Thus, the optimal solution is

$$x_0^* = 16.04, \quad \mathbf{x}^* = (0.32,\ 0.2) \quad \mathbf{y}^* = (0, 1, 0, 0, 0, 1)\,.$$

Notes: The solution of discrete problems containing the single continuous variable x_0 takes place as was described in the first solution, using Minkowski's decomposition.

Initially, the first method seems to be simpler as only one discrete and one continuous problem need to be solved. It is not so however—even for problems of medium size—because the number of vertices of polyhedron P, that is, the number of extremal directions of cone C, is very large even for relatively small problems.

On the other hand, in the second method all discrete problems differ from the previous one only by a single inequality. Because of this, special algorithms need be developed which do not start each time from the beginning. As can be seen in the literature, only a reasonable number of extremal directions occur in practice, so this approach is considered as practically feasible even for larger problems.

CHAPTER 8

MODIFIED FILTER METHOD

The method described in the present chapter originates from the filter method of Balas (1967). The notion of a filter, which is nothing other than an s-constraint, similar to those described in Chapter 5, is taken from Balas' paper. Otherwise the algorithm itself and most of the proofs are quite different. The treatment of the chapter is closely related to that of Chapter 2, but there is also a slight deviation in order to make the discussion more suggestive.

The main purpose of the algorithm is to demonstrate the possibility to use single-branch enumeration and the branch-and-bound method in a common framework. An effort is made to limit the number of branches otherwise the growing core requirement would be prohibitive. Further suggestions are made for memory saving, following the description of the algorithm.

The method may be applied to mixed integer problems by using the Benders decomposition detailed in Chapter 7.

8.1 Problem formulation

Let us consider the following problem:

Problem (P):

$$\min \mathbf{c}^T\mathbf{x} \tag{8.1}$$

$$\mathbf{A}\mathbf{x} \geqq \mathbf{b} \tag{8.2}$$

$$0 \leqq x_j \leqq 1 \qquad (j = 1, 2, \ldots, n) \tag{8.3}$$

$$x_j \text{ integer} \qquad (j = 1, 2, \ldots, n)\,. \tag{8.4}$$

As was seen in Chapter 2, without confining the generality we can suppose that $\mathbf{c} \geqq \mathbf{0}$. $\mathbf{A}$ is an $m \times n$ matrix. For simplicity, all coefficients are supposed as being integer. A vector $\mathbf{x}$ satisfying requirements (8.3) and (8.4) is called a solution. If the set of inequalities (8.2) are also satisfied by $\mathbf{x}$, then it is a feasible solution.

Problem (P') is defined as the continuous version of Problem (P): (8.1)–(8.3).

Problem (D') is the dual of Problem (P') in the sense of linear programming

$$\max \mathbf{b}^T\mathbf{u} - \mathbf{e}^T\mathbf{v} \tag{8.5}$$

$$\mathbf{A}^T\mathbf{u} - \mathbf{v} \leqq \mathbf{c} \tag{8.6}$$

$$\mathbf{u} \geqq \mathbf{0}, \quad \mathbf{v} \geqq \mathbf{0}, \tag{8.7}$$

where

$$\mathbf{e}^T = (1, 1, \ldots, 1)\,.$$

Constraints (8.2)–(8.3) define a bounded region. Therefore if Problem (P') has a feasible solution, then it has an optimal solution ($\hat{\mathbf{x}}$) as well. In this case, according to the duality theorem of linear programming, Problem (D') also has an optimal solution which is denoted by ($\hat{\mathbf{u}}$, $\hat{\mathbf{v}}$). Let us now form a new problem having only a single inequality instead of system (8.2). The nonnegative multipliers of inequalities (8.2) are the components of vector $\hat{\mathbf{u}}$

Problem $F(\hat{u})$:

$$\min \mathbf{c}^T\mathbf{x} \tag{8.8}$$

$$\mathbf{a}^T\mathbf{x} \geqq b_0 \tag{8.9}$$

$$0 \leqq x_j \leqq 1 \qquad (j = 1, 2, \ldots, n) \tag{8.10}$$

$$x_j \text{ integer} \qquad (j = 1, 2, \ldots, n) \tag{8.11}$$

where

$$\mathbf{a}^T = \hat{\mathbf{u}}^T\mathbf{A} \quad \text{and} \quad b_0 = \hat{\mathbf{u}}^T\mathbf{b}\,.$$

As inequality (8.9) is a consequence of inequality system (8.2), all feasible solutions of Problem (P) satisfy the constraints of Problem $F(\hat{\mathbf{u}})$. This implies that if the optimal solution of Problem $F(\hat{\mathbf{u}})$ satisfies the inequality system (8.2), then it is also an optimal solution for Problem (P). Inequality (8.9) is called a filter because it filters out a large part of solutions nonfeasible for Problem (P). It plays exactly the same role as did the s-constraints in Chapter 5. The only difference being in the determination of the multipliers u.

Problem $F'(\hat{\mathbf{u}})$ is the continuous version of Problem $F(\hat{\mathbf{u}})$: (8.8)–(8.10).

Problem $K'(\hat{\mathbf{u}})$ is the dual of Problem $F'(\hat{\mathbf{u}})$

$$\max b_0u_0 - \mathbf{e}^T\mathbf{v} \tag{8.12}$$

$$\mathbf{a}u_0 - \mathbf{v} \leqq \mathbf{c} \tag{8.13}$$

$$u_0 \geqq 0, \quad \mathbf{v} \geqq \mathbf{0}, \tag{8.14}$$

where

$$a = \mathbf{A}^T\mathbf{u}, \quad b_0 = \mathbf{b}^T\hat{\mathbf{u}} \quad \text{and} \quad \mathbf{e}^T = (1, 1, \ldots, 1)\,.$$

The choice of multipliers $\mathbf{u}$ is justified by the following theorem.

Theorem 8.1 (i) *If* $\hat{\mathbf{x}}$ *and* $(\hat{\mathbf{u}}, \hat{\mathbf{v}})$ *are optimal solutions of Problem* (P') *and* (D') *respectively, then* $\hat{\mathbf{x}}$ *is also an optimal solution of Problem* $F'(\hat{\mathbf{u}})$.

(ii) $\mathbf{c}^T\mathbf{x}$ *is not larger than the optimum of Problem* $F'(\hat{\mathbf{u}})$.

(iii) *There exists at least one pair of feasible solutions* $\mathbf{x}'$ *and* $(\hat{\mathbf{u}}', \hat{\mathbf{v}}')$ of *Problems* (P') *and* (D') respectively, *in such a way that* $x_j' = 1$ *implies* $x_j' = 1$ *and* $x_j' = 0$ *implies* $\tilde{x}_j' = 0$ *for any optimal solution* $\tilde{\mathbf{x}}$ *of Problem* $F'(\mathbf{u}')$.

Proof: (i) Problem $K'(\hat{\mathbf{u}})$ can be considered as looking for the solutions of type $(u_0, \hat{\mathbf{u}}, \mathbf{v})$ for Problem (D'). This obviously cannot be better than the optimal solution chosen from among all feasible vectors $(\mathbf{u}, \mathbf{v})$. Thus the optimum of Problem $K'(\hat{\mathbf{u}})$ is less than or equal to the optimum of Problem (D'). On the other hand, the choice

$$u_0^* = 1, \qquad \mathbf{v}^* = \hat{\mathbf{v}}$$

results in a feasible solution of Problem $K'(\hat{\mathbf{u}})$ and its objective function value is equal to the optimum of Problem (D'), thus $(1, \hat{\mathbf{v}})$ is an optimal solution of Problem $K'(\hat{\mathbf{u}})$.

Applying the duality theorem of linear programming for Problems (P') and (D'), we obtain

$$\mathbf{c}^T\hat{\mathbf{x}} = \mathbf{b}^T\hat{\mathbf{u}} - \mathbf{e}^T\hat{\mathbf{v}}\,. \tag{8.15}$$

Vector $\hat{\mathbf{v}}$ is obviously a feasible solution of problem $F'(\hat{\mathbf{u}})$. According to equation (8.15), the value $\mathbf{c}^T\hat{\mathbf{x}}$ is equal to the optimum of Problem $K'(\hat{\mathbf{u}})$ and thus, by the duality theorem, it is equal to the optimum of Problem $F'(\hat{\mathbf{u}})$ – which proves Statement (i).

Assertion (ii) follows from the fact that Problem $F(\hat{\mathbf{u}})$ has a feasibility region which is part of the feasibility set of Problem $F'(\hat{\mathbf{u}})$.

Statement (iii) may be seen with the help of strong complementary slackness for linear programming [see e.g. Hadley (1962), Prékopa (in press)]. This means, in our case, that there exists a pair of optimal solutions $\mathbf{x}'$ and $(\mathbf{u}, \mathbf{v}')$ for Problems (P') and (D') respectively, in such a way that

$$x_j' = 1 \text{ implies } A_j^T u' - v_j' = c_j \text{ and } v_j > 0$$

$$x_j' = 0 \text{ implies } A_j^T u' - v_j' < c_j \text{ and } v_j' = 0$$

$$0 < x_j < 1 \text{ implies } A_j^T u' - v_j' = c_j \text{ and } v_j' = 0,$$

where A_j denotes the jth column vector of matrix $\mathbf{A}$. As $\mathbf{A}_j^T\mathbf{u}' = a_j$, the coefficients of inequality (8.9), the above implications may be formulated as follows

$$x_j' = 1, \quad \text{if} \quad a_j > c_j \tag{8.16}$$

$$x_j' = 0, \quad \text{if} \quad a_j < c_j \tag{8.17}$$

$$0 < x_j' < 1, \quad \text{if} \quad a_j = c_j \tag{8.18}$$

Let us now choose any optimal solution $\tilde{\mathbf{x}}$ of problem $F'(\mathbf{u}')$. On the basis of Lemmata 1.1 and 1.2 of Chapter 1 and using the fact that an optimal solution with properties (8.16)–(8.18) exists, it is easy to see that

$$\tilde{x}_j = 1, \quad \text{only if} \quad a_j \geqq c_j$$

$$\tilde{x}_j = 0, \quad \text{only if} \quad a_j \leqq c_j$$

$$0 < \tilde{x}_j < 1, \quad \text{only if} \quad a_j = c_j$$

which completes the proof of the theorem.

Notes: According to part (i) of the theorem, Problem $F(\hat{\mathbf{u}})$ need never be solved. Part (ii) shows that any vector $\mathbf{x}$ with $\mathbf{c}^T\mathbf{x} < \mathbf{c}^T\hat{\mathbf{x}}$ may be ignored. Finally, part (iii) helps in solving strongly degenerate problems. It says that there exists a core of variables taking given integer values in each optimal solution of problem $F'(\mathbf{u}')$.

8.2 Optimal indexing

After solving Problems (P') and (D'), the optimal solutions $\hat{\mathbf{x}}$ and $(\hat{\mathbf{u}}, \hat{\mathbf{v}})$ respectively, are used to create Problem $F(\hat{\mathbf{u}})$ which plays the role of a filter. As the order of variables is important, a so-called optimal indexing will be introduced. Let us use the notation

$$H_n = \{1, 2, \ldots, n\}$$

and j_k denotes the serial number of variable x_k in the new ordering, which is defined by the following rules.

For any $r, s \in H_n$, $j_r < j_s$, if one of the three conditions holds

$$a_r > 0, \quad a_s \leqq 0 \tag{8.19}$$

$$a_r > 0, \; a_s > 0, \; \frac{c_r}{a_r} < \frac{c_s}{a_s} \tag{8.20}$$

$$a_r \leqq 0, \; a_s \leqq 0 \quad a_r > a_s . \tag{8.21}$$

Several orderings may satisfy the above restrictions, any of them may be accepted as a new ordering.

In the method we are looking for feasible solutions. The above ordering may be interpreted accordingly.

In the remaining part of the chapter the variables are supposed as being reordered to satisfy rules (8.19)–(8.21). (See also notes at the end of the chapter.)

8.3 Solution-tree and pseudo-solution tree

The definitions of Chapter 2 will be used and will be given suggestive meanings. First of all, we shall define the solution-tree, i.e. a directed graph each vertex of which represents a solution. (An n-dimensional vector of 0–1 components.)

Now we have to define the edges of this graph. There is an edge from vertex x to vertex y, if

$$x_1 = y_1, \ldots, x_{r-1} = y_{r-1}, \; x_r = 0, \; y_r = 1$$
$$x_{r+1} = y_{r+1} = \ldots = x_n = y_n = 0, \tag{8.22}$$

where

$$1 \leqq r \leqq n .$$

The solution tree for $n = 4$ may be seen in Figure 8.1. It is clear that each solution has been obtained exactly once. The unique immediate ascendent of any element (except $\mathbf{x} = \mathbf{0}$) may be obtained by changing the last 1 to 0. Another notation may be used for the vertices of the tree. If the zeros at the end of each vector are omitted, the picture will be clearer. At the same time we use the notation i for $x_i = 1$ and $\bar{i}$ for $x_i = 0$. Now Figure 8.2 shows the solution-tree using the new notation. A new interpretation may be given to this form of the solution tree. At the beginning, each variable is free and free variables are considered to take 0 value. Now let us form the next n points of the graph by fixing one of the variables at value 1 (1 is assigned to different variables at each vertex). The variables preceding it will be fixed at 0. The variables following the newly assigned ones will remain free and are considered to be 0 temporarily. The same rule is repeated for each new vertex, with the difference that only the free variables are considered for further fixings.

Figure 8.2 may also be considered as a tree of ordered pseudo-solutions.

The definitions of Chapter 2 will be used in the following discussion. As in Chapter 6, ψ^0 will denote the root of pseudo-solution ψ, that is, the solution obtained by fixing the free variables at 0 value. The root of an ordered pseudo-solution is interpreted as the root of the corresponding not ordered one:

$$\phi^0 = \{\phi\}^0 .$$

A pseudo-solution of one element (all variables are fixed) is considered identical with this single element.

Now, some definitions and lemmata will be formulated for the ordered pseudo-solution tree. The word "ordered" may be deleted for simplicity without the danger of misunderstanding because only one such tree will be used.

Definition 8.1: *A pseudo-solution tree is a graph $G = (V, E)$, where V is the set of vertices consisting of ordered pseudo-solutions with the following properties:*

(i) *The fixed variables retain the order of optimal indexing, defined in Section* 8.2.

(ii) *If variable x_r is fixed, then so are variables $x_1, x_2, \ldots, x_{r-1}$.*

(iii) *The last fixed variable has the value* 1.

(iv) $(\emptyset) \in V$. *(The empty ordered pseudo-solution is in V.)*

E, the set of directed edges, is defined as follows

$(\phi_1, \phi_2) \in E$ if $\phi_2 = (\phi_1, x_{r+1} = \delta_{r+1}, \ldots, x_{r+s-1} = \delta_{r+s-1}, x_{r+s} = 1)$,

where r is the number of variables fixed in the ordered pseudo-solution ϕ_1 ($r = 0, 1, \ldots, n-1$; $s = 1, 2, \ldots, n-r$).

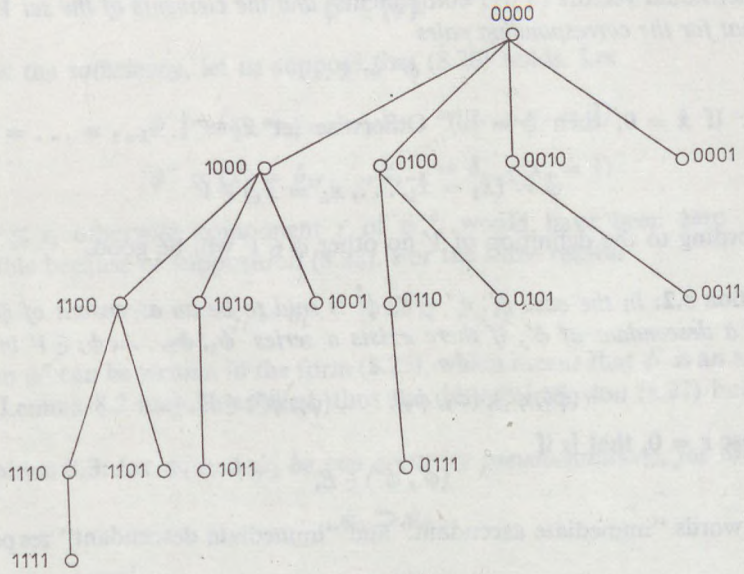


Fig. 8.1

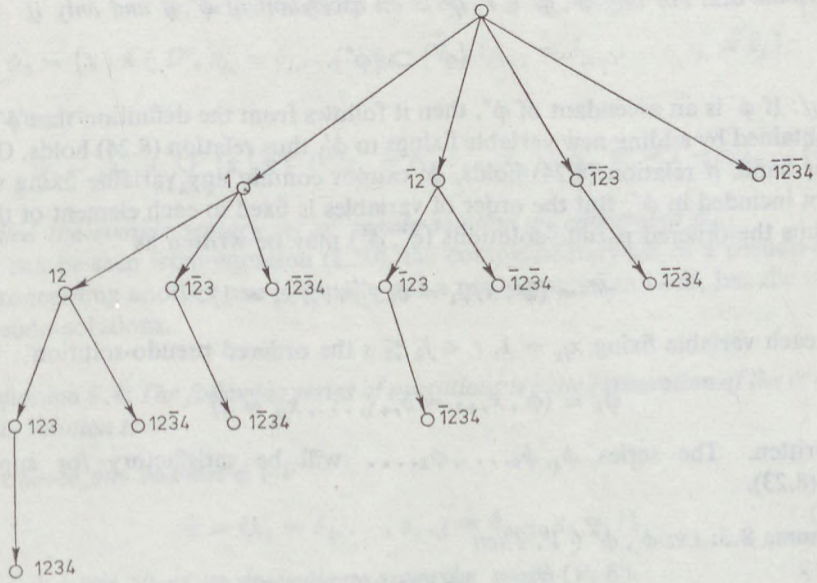


Fig. 8.2

The graph (V, E) is obviously a tree, i.e. connected and contains no loop. We now formulate the equivalence of the solution tree and the pseudo-solution tree defined above.

Lemma 8.1: *There exists a one-to-one correspondence between the solutions* $\hat{\mathbf{x}}$ (n-dimensional vectors of 0–1 components) *and the elements of the set* V *in such a way that for the corresponding pairs*

$$\hat{\phi}^0 = \hat{\mathbf{x}}\,.$$

Proof: If $\hat{\mathbf{x}} = \mathbf{0}$, then $\hat{\phi} = (\emptyset)$. Otherwise let $\hat{x}_k = 1, \hat{x}_{k+1} = \ldots = \hat{x}_n = 0$, then

$$\hat{\phi} = (x_1 = \hat{x}_1, \ldots, x_k = \hat{x}_k) \in V$$

and according to the definition of V no other $\phi \in V$ will be good.

Definition 8.2: *In the case* $\phi', \phi'' \in V$, ϕ' *is said to be an ascendant of* ϕ'' *or* ϕ'' *is called a descendant of* ϕ', *if there exists a series* $\phi_1, \phi_2, \ldots, \phi_r \in V$ *in such a way that*

$$(\phi', \phi_1), (\phi_1, \phi_2), \ldots, (\phi_r, \phi'') \in E\,. \tag{8.23}$$

In the case $r = 0$, that is if

$$(\phi', \phi'') \in E,$$

then the words "immediate ascendant" and "immediate descendant" respectively are used.

Lemma 8.2: *For any* $\phi', \phi'' \in V$, ϕ' *is an ascendant of* ϕ'' *if and only if*

$$\{\phi'\} \supset \{\phi''\}. \tag{8.24}$$

Proof: If ϕ' is an ascendant of ϕ'', then it follows from the definition that ϕ'' may be obtained by adding new variable fixings to ϕ', thus relation (8.24) holds. On the other hand, if relation (8.24) holds, ϕ' cannot contain any variable fixing which is not included in ϕ''. But the order of variables is fixed in each element of the set V, thus the ordered pseudo-solutions (ϕ', ϕ'') may be written as

$$\phi'' = (\phi', x_{r+1} = \delta_{r+1}, \ldots, x_s = 1)\,. \tag{8.25}$$

For each variable fixing $x_{j_k} = 1$, $r < j_k \leqq s$ the ordered pseudo-solution

$$\phi_k = (\phi', x_{r+1} = \delta_{r+1}, \ldots, x_{j_k} = 1) \tag{8.26}$$

is written. The series $\phi_1, \phi_2, \ldots, \phi_k, \ldots$ will be satisfactory for supposition (8.23).

Lemma 8.3: *Let* $\phi', \phi'' \in V$. *Then*

$$\{\phi'\} \supset \{\phi''\} \tag{8.27}$$

if and only if

$$\phi''^0 \in \{\phi'\}. \tag{8.28}$$

Proof: The necessity of the condition is obvious because for any ordered pseudo-solution ϕ

$$\phi^0 \in \{\phi\}.$$

To show the sufficiency, let us suppose that (8.28) holds. Let

$$\phi' = (x_1 = \gamma_1, \ldots, x_{r-1} = \gamma_{r-1}, x_r = 1)$$

$$\phi'' = (x_1 = \delta_1, \ldots, x_{s-1} = \delta_{s-1}, x_s = 1).$$

Then $r \leqq s$, otherwise component r of ϕ''^0, would have been zero which is impossible because of supposition (8.28). For the same reason

$$\gamma_i = \delta_i \qquad (i = 1, \ldots, r).$$

But then ϕ'' can be written in the form (8.25), which means that ϕ' is an ascendant of ϕ''. Lemma 8.2 may be applied, thus the desired inclusion (8.27) holds.

Definition 8.3: *Let* ψ_1 and ψ_2 *be two arbitrary pseudo-solutions, for which*

$$\psi_1 \supset \psi_2$$

That is, in general

$$\psi_1 = \{\mathbf{x} \mid \mathbf{x} \in D^n, x_{j_1} = \delta_{j_1}, \ldots, x_{j_k} = \delta_{j_k}\} \tag{8.29}$$

$$\psi_2 = \{\mathbf{x} \mid \mathbf{x} \in D^n, x_{j_1} = \delta_{j_1}, \ldots, x_{j_k} = \delta_{j_k}, x_{j_{k+1}} = \delta_{j_{k+1}}, \ldots, x_{j_t} = \delta_{j_t}\}.$$

Then the set

$$\psi_1 - \psi_2 = \bigcup_{r=k+1}^{t} \{\mathbf{x} \mid \mathbf{x} \in D^n, x_{j_1} = \delta_{j_1}, \ldots, x_{j_k} = \delta_{j_k}, x_{j_r} = 1 - \delta_{j_r}\}. \tag{8.30}$$

is called the complementary set of pseudo-solution ψ_2 *concerning* ψ_1.

As can be seen from equation (8.30), the complementary set of a pseudo-solution concerning another one is usually not a pseudo-solution itself, but the union of pseudo-solutions.

Definition 8.4: *The following series of operations is called* **truncation** *of the ordered pseudo-solution tree.*

(i) *Choose any element* $\phi \in V$

$$\phi = (x_1 = \delta_1, \ldots, x_{r-1} = \delta_{r-1}, x_r = 1) \tag{8.31}$$

and delete all of its descendants from the graph (V, E).

(ii) *Choose an arbitary ordered pseudo-solution, such that* $\{\phi'\} \subset \{\phi\}$, *that is,*

$$\phi' = (x_1 = \delta_1, \ldots, x_{r-1} = \delta_{r-1}, x_r = 1, x_{j_r+1} = \delta_{j_r+1}, \ldots, x_{j_{r+s}} = \delta_{j_{r+s}}), \quad (8.32)$$

and substitute ϕ *by* ϕ'. ($\phi \in V$ is not required!)

(iii) *Starting from ordered pseudo-solution* ϕ' *build up the remaining part of the graph as in Definition 8.1 considering the variables with subscripts*

$$1 \leqq j \leqq n \ (i \neq 1, \ldots, r, j_{r+1}, \ldots, j_{r+s}) . \quad (8.33)$$

The graph so obtained is denoted by (V_1, E_1).

Definition 8.5: *If a finite number of truncations is executed on the ordered pseudo-solution tree* (V, E) *(each truncation is applied to the result of the previous one), the final graph* (V_k, E_k) *is called the* **truncated tree of ordered pseudo-solutions** *or shortly* **truncated tree.**

Notes:

1° Truncated trees may also be obtained in another way. The graph (V, E) is built up vertex by vertex, starting from $\phi = (\emptyset)$ and considering only the vertices connected to the part of the graph already accepted. After including a new vertex in the accepted part, some of the variables still free in this ordered pseudo-solution may be fixed at 0 or 1 and joined after the existing variable fixings. Then the process is continued (considering only the free variables) as described in definition 8.1. In such a way another definition, equivalent to that of 8.5, may be obtained.

2° It is clear, especially after the preceeding note, that the purpose of truncation is that consequences of variable fixings, according to the constraints of the problem, may be taken into account. Truncation reduces the size of the pseudo-solution tree and thus speeds up the algorithm.

3° Obviously no truncated tree contains a loop, thus the use of word "tree" is correct.

Example: If in the four dimensional pseudo-solution tree $x_1 = 1$ implies $x_3 = 0$ and $x_2 = 1$, furthermore $x_1 = 0$ implies $x_2 + x_3 + x_4 = 2$, then the pseudo-solution tree of Figure 8.2 may be substituted by the truncated tree of Figure 8.3.

Definition 8.6: *Let* (V', E') *be a truncated tree, and let*

$$T(V') = \bigcup_{\phi \in V'} \{\phi\} .$$

Furthermore S *denotes the set of feasible solutions of Problem* (P). *The set* H *is called a covering system of set* V' *concerning Problem* (P) *if* $H \subset V'$ *and*

$$\bigcup_{\phi \in H} \{\phi\} \supset S \cap T(V').$$

The following statement is an obvious consequence of the definition.

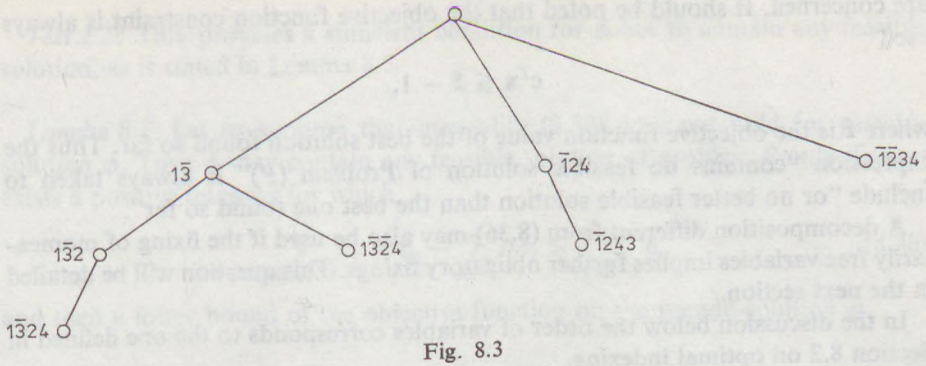


Fig. 8.3

Lemma 8.4: *If set $H \subset V'$ is a covering system of the truncated tree (V', E'), concerning Problem (P), and ϕ^0 is not a feasible solution of Problem (P), then — denoting the immediate descendants of ϕ by $\phi_1, \ldots, \phi_r$ — the set*

$$(H - \phi) \cup \{\phi_1, \ldots, \phi_r\}$$

is also a covering system.

8.4 Use of the branch-and-bound principle

The application of the branch-and-bound principle, discussed in Chapter 3 in detail, follows quite naturally from the foregoing discussion.

It is easy to obtain a lower bound for the feasible solutions of Problem (P) in the pseudo-solution $\{\phi\}$. Namely, $\mathbf{c} \geqq \mathbf{0}$ implies that no better objective function value can be achieved than that of ϕ^0. If ϕ^0 is a feasible solution, then this is also the best one in ϕ, thus ϕ is fathomed. In the opposite case ϕ is decomposed according to Lemma 8.4.

A truncated tree is to be built up gradually. Let

$$\phi = (x_{j_1} = \delta_{j_1}, \ldots, x_{j_r} = \delta_{j_r}) \tag{8.34}$$

be an ordered pseudo-solution of this truncated tree, which is not yet decomposed. Furthermore, denote the subscripts of the free variables (in natural order), that is, $1 \leqq j \leqq n$, $j \neq j_1, \ldots, j_r$, by

$$k_1, k_2, \ldots, k_{n-r}. \tag{8.35}$$

Then ϕ is substituted in the covering system by

$$\phi_k = (\phi, x_{k_1} = 0, \ldots, x_{k_{t-1}} = 0, x_{k_t} = 1) \quad (t = 1, 2, \ldots, n - r). \tag{8.36}$$

The covering system always consists of the pseudo-solutions not yet decomposed and not yet proved to be empty, as far as the feasible solutions of Problem (P)

are concerned. It should be noted that the objective function constraint is always used

$$\mathbf{c}^T\mathbf{x} \leqq \hat{z} - 1,$$

where $\hat{z}$ is the objective function value of the best solution found so far. Thus the expression "contains no feasible solution of Problem (P)" is always taken to include "or no better feasible solution than the best one found so far".

A decomposition different from (8.36) may also be used if the fixing of momentarily free variables implies further obligatory fixings. This question will be detailed in the next section.

In the discussion below the order of variables corresponds to the one defined in Section 8.2 on optimal indexing.

8.5 Tests

The purpose of tests is to show that in the examined pseudo-solution either no feasible solution exists or some of the free variables should be fixed at 1, others at 0, otherwise no feasible solution exists. In the latter case the obligatory fixings are introduced and a truncation operation of the pseudo-solution tree is executed. Obviously, the definition of feasible solution always includes the objective function constraint. (See the above note.)

In the present chapter two groups of tests will be used. One of them is applied only for the filter (F tests) the other for the original constraints (P tests). The F tests usually require less time and provide fewer results than the P tests.

The tests are formulated for the general ordered pseudo-solution (8.34) which is repeated here for convenience

$$\phi = (x_{j_1} = \delta_{j_1}, \ldots, x_{j_r} = \delta_{j_r}).$$

The set of fixed variables is denoted by

$$J = \{j_1, \ldots, j_r\}$$

and the notation

$$H_t = \{1, 2, \ldots, t\}$$

will also be used.

Test F1: If ϕ^0, the root of pseudo-solution ϕ does not satisfy filter (8.9), i.e. the inequality

$$\sum_{j \in J} a_j \delta_j \geqq b_0 \tag{8.37}$$

does not hold, then ϕ^0 is not a feasible solution of Problem (P).

If Test $F1$ shows the infeasibility of ϕ^0, i.e. inequality (8.37) does not hold, then we apply

Test F2: This provides a sufficient condition for ϕ not to contain any feasible solution, as is stated in Lemma 8.5.

Lemma 8.5: Let us suppose that inequality (8.37) does not hold for pseudo-solution ϕ. Then ϕ may contain any feasible solution of problem P only if there exists a positive integer t for which

$$\sum_{s \in H_{t-1}-J} a_s < b_0 - \sum_{j \in J} a_j \delta_j \leqq \sum_{s \in H_t - J} a_s \tag{8.38}$$

and then a lower bound of the objective function on the pseudo-solution ϕ

$$\sum_{j \in J} c_j \delta_j + \sum_{s \in H_{t-1}-J} c_s + \frac{c_t}{a_t} \left(b_0 - \sum_{j \in J} a_j - \sum_{s \in H_{t-1}-J} a_s \right). \tag{8.39}$$

Proof: As ϕ does not satisfy inequality (8.37), the left hand side inequality in (8.38) holds for $t = 1$. We have to show that if

$$\sum_{s \in H_t - J} a_s < b_0 - \sum_{j \in J} a_j \delta_j \tag{8.40}$$

for $t = 1, 2, \ldots, n$, then ϕ contains no feasible solution. Let us suppose on the contrary, that x is a feasible solution. Then naturally it satisfies the filter

$$\sum_{j \in J} a_j \delta_j + \sum_{s \in H_n - J} a_s \tilde{x} \geqq b_0. \tag{8.41}$$

Then define solution $\mathbf{x}'$ by the following rules

$$x'_j = \begin{cases} x_j & \text{if} \quad j \in J \quad \text{or} \quad a_j > 0 \\ 0 & \text{otherwise} \end{cases} \tag{8.42}$$

and let

$$q = \max \{ j \mid j \notin J, \quad a_j > 0 \} \tag{8.43}$$

Then, according to the optimal indexing of section 8.2,

$$\sum_{s \in H_n - J}^{n} a_s \tilde{x}_s \leqq \sum_{s \in H_n - J} a_s x'_s = \sum_{s \in H_q - J}^{q} a_s. \tag{8.44}$$

Inequalities (8.41) and (8.44) together give the desired inequality (8.38).

In order to show that expression (8.39) is a lower bound of ϕ, let us suppose that inequalities (8.38) hold for an integer t. Let $\tilde{\mathbf{x}} \in \{\phi\}$ be a feasible solution. Then inequality (8.41) holds and, as a consequence of condition (8.38),

$$a_t > 0.$$

Thus, according to the ordering of variables

$$a_s > 0 \qquad s \in H_t - J \tag{8.45}$$

using subscript q defined by equation (8.43)

$$\sum_{j \in J} a_j \delta_j + \sum_{s \in H_q - J}^{q} a_s \tilde{x}_s \geqq b_0 . \tag{8.46}$$

The objective function may be divided into three parts

$$\sum_{j \in J} c_j \delta_j + \sum_{s \in H_q - J} c_s \tilde{x}_s + \sum_{s \in H_n - H_q - J} c_s \tilde{x}_s \tag{8.47}$$

$\mathbf{c} \geqq \mathbf{0}$ implies, that

$$\sum_{s \in H_n - H_q - J} c_s \tilde{x}_s \geqq 0 . \tag{8.48}$$

Furthermore, according to the ordering principle (8.20) and inequality (8.46)

$$\sum_{s \in H_q - J} c_s \tilde{x}_s \geqq \sum_{s \in H_{t-1} - J} c_s + \frac{c_t}{a_t} \left(b_0 - \sum_{j \in J} a_j \delta_j - \sum_{s \in H - J} a_s \right) . \tag{8.49}$$

This inequality may be obtained by using the result of Lemma 1.2 of Chapter 1. The objective function value (8.47) may be estimated by using inequalities (8.48) and (8.49) and thus, the lower bound (8.39) is correct.

Note. It is possible, that the filter in the algorithm becomes ineffective in a pseudo-solution, that is,

$$\sum_{j \in J} a_j \delta_j \geqq b_0 .$$

In this case either we calculate another filter for this branch or we use another lower bound calculation. A very simple method would be, for example, to calculate a lower bound of ϕ' ($\{\phi'\} \subseteq \{\phi\}$) if ϕ'^0 is not a feasible solution

$$\sum_{j \in J'} a_j \delta_j + c_k ,$$

where J' is the set of fixed variables in ϕ' and

$$c_k = \min \{ c_j \mid j \in J' \} .$$

A more sophisticated method can easily be designed using one or several of the constraints not satisfied by solution ϕ^0.

Test P1: Similarly to Test *F1* this test refers to the root ϕ^0 of the just examined pseudo-solution ϕ. If the inequalities

$$\sum_{j \in J} a_{ij} \delta_j \geqq b_i \quad (i = 1, 2, \ldots, m) \tag{8.50}$$

hold, then a new feasible solution has been found.

If ϕ^0 is not a feasible solution, then all the descendants of ϕ^0 are checked for feasibility:

Test P2: Let us construct the inequalities

$$\sum_{j \in J} a_j \delta_j + \sum_{j \in H_n - J}^{n} a_{ij}^+ \geqq b_i \quad (i = 1, \ldots, m) \tag{8.51}$$

where the + sign in the superscript denotes the positive part of the coefficient

$$d^+ = \begin{cases} d & \text{if } d > 0 \\ 0 & \text{if } d \leqq 0. \end{cases}$$

If at least one of the inequalities (8.51) do not hold, then the pseudo-solution $\{\phi\}$ contains no feasible solution. This branch may thus be closed.

In the opposite case the next test is applied in order to find obligatory variable fixings.

Test P3: Test *P2* is applied to the pseudo-solutions

$$\phi_s' = (\phi, x_s = 0) \qquad (s \notin J) \tag{8.52}$$

and

$$\phi_s'' = (\phi, x_s = 1) \qquad (s \notin J). \tag{8.53}$$

Let us denote the set of subscripts for which ϕ_s' and ϕ_s'' contain no feasible solution, by F_1 and F_0 respectively. Then the variable fixings

$$x_s = 1 \ (s \in F_1) \quad \text{and} \quad x_s = 0 \ (s \in F_0)$$

are obligatory in the pseudo-solution ϕ.

Obviously, if

$$F_1 \cap F_0 \neq \emptyset, \tag{8.54}$$

then ϕ contains no feasible solution. If

$$F_1 \cup F_0 = \emptyset, \tag{8.55}$$

then no information is obtained from the test, and finally if

$$F_1 \cap F_0 = \emptyset \quad \text{and} \quad F_1 \cup F_0 \neq \emptyset, \tag{8.55}$$

then the corresponding variable fixings should be executed as a truncation of the pseudo-solution tree, as described in Definition 8.4. This corresponds to the tying of the variables in Chapter 2.

8.6 Description of the algorithm

As was already mentioned, the method is based on the branch-and-bound principle but a filter constraint and several tests are applied to reduce the pseudo-solutions. Furthermore, the optimal indexing is used to decrease the number of branches formed at each separation of solutions. This is possible because the lower bounds for feasible solutions are monotone increasing for the immediate descendants of any ordered pseudo-solution if they are formed according to the variable ordering introduced in Section 8.2.

The symbol π will denote the set of subscripts of all ordered pseudo-solutions that should be taken into account, i.e. the ones not yet decomposed and not yet shown to be empty as far as feasible solutions are concerned. (Naturally the objective function constraint is also used, thus only feasible solutions better than the best one obtained so far are considered.) As the objective function coefficients $\mathbf{c} \geqq \mathbf{0}$, all lower bounds are also nonnegative.

The sign $:=$ will be used for the following purpose. The same variable or set may stand on both sides of this sign, but the left hand side denotes its new value and the right hand side contains the old value. For example if π denotes a certain set of positive integers, then

$$\pi := \pi \cup \{8\} - \{3\}$$

means that the number 8 is added to set π and the number 3 is deleted. The new set is again called π.

A descendant indicator α_t is assigned to each generated ordered pseudo-solution ϕ_t. $\alpha_t = 0$ means that the immediate ascendant of ϕ_t contains at least one ordered pseudo-solution which comes later than ϕ_t in the ordering and it also means that none of these pseudo-solutions has already been generated. In other words the lower bound $z_t = z(\phi_t)$ of the objective function of ϕ_t also stands for other pseudo-solutions not yet generated. In the opposite case $\alpha_t = 1$.

* * *

The algorithm for solving Problem (P) may be summarized in the following steps:

Step 1: Solve Problem (P'). If it has no solution, then Problem (P) has no solution. Otherwise continue at

Step 2: Denote an optimal solution of Problem (P') by $\hat{\mathbf{x}}$ and the corresponding objective function value by $\hat{z}$. If all components of $\hat{\mathbf{x}}$ are integer, then it is also the optimal solution of Problem (P).

Step 3: If $\hat{\mathbf{x}}$ has noninteger components, then solve Problem (D') and denote its optimal solution by $(\hat{\mathbf{u}}, \hat{\mathbf{v}})$. Form the corresponding Problem $F(\hat{\mathbf{u}})$.

Step 4: A new order of variables is introduced according to rules (8.19)–(8.21).

Step 5: Initialization. Let

$$\pi = \emptyset, \quad t = 0, \qquad \phi_0 = \emptyset, \quad \alpha_0 = 1 .$$

Go to Step 8.

Step 6: If $\pi = \emptyset$, then Problem (P) has no solution. Otherwise determine

$$\min_{i \in \pi} z(\phi_i) = z(\phi_{i_0}),$$

and let

$$z_0 = z(\phi_{i_0}), \quad \phi_0 = \phi_{i_0}, \quad \alpha_0 = \alpha_{i_0},$$
$$\pi := \pi - \{i_0\}$$

If ϕ_0^0, the root of ϕ_0, is a feasible solution, then it is also the optimal solution of Problem (P). In the opposite case

Step 7: If $\alpha_0 = 1$ go to Step 8, otherwise form a new ordered pseudo-solution ϕ_{t+1} from

$$\phi_0 = (x_{j_1} = \delta_{j_1}, \ldots, x_{j_{r-1}} = \delta_{j_{r-1}}, x_{j_r} = 1)$$

such that

$$\phi_{t+1} = (x_{j_1} = \delta_{j_1}, \ldots, x_{j_{r-1}} = \delta_{j_{r-1}}, x_{j_r} = 0, x_s = 1)$$

where

$$s = \min \{j \mid j \in H_n - J\} .$$

Calculate z_{t+1} according to Lemma 8.5 and let $\alpha_{t+1} = 1$, if all variables are fixed in ϕ_{t+1}, otherwise $\alpha_{t+1} = 0$. Furthermore

$$\pi := \pi \cup \{t + 1\} \qquad t := t + 1 .$$

Step 8: Apply tests $F2$ and $P2$. If one of them shows the infeasibility of ϕ_0, then go to Step 6. Otherwise apply test $P3$ and form the sets F_1 and F_0 for obligatory variable fixings.

Step 9: If

$$F_1 \cap F_0 \neq \emptyset.$$

then go to Step 6. If

$$F_1 \cup F_0 = \emptyset$$

then go to Step 10. Finally if

$$F_1 \cap F_0 = \emptyset \quad \text{and} \quad F_1 \cup F_0 \neq \emptyset,$$

then go to Step 11.

Step 10: Form a descendant of the present ordered pseudo-solution

$$\phi_0 = (x_{j_1} = \delta_{j_1}, \ldots, x_{j_r} = 1)$$
$$\phi_{t+1} = (x_{j_t} = \delta_{j_t}, \ldots, x_{j_r} = 1, \; x_s = 1),$$

where

$$s = \min \{j \mid j \in H_n - J\} .$$

Let $\alpha_{t+1} = 1$, if all variables are fixed in ϕ_{t+1}, otherwise let $\alpha_{t+1} = 0$.
Go to *Step* 12.

Step 11: Form a new descendent of the present pseudo-solution

$$\phi_{t+1} = (\phi, x_{p_1} = 1, \ldots, x_{p_r} = 1, x_{q_1} = 0, \ldots, x_{q_s} = 0)$$

where

$$F_1 = \{p_1, \ldots, p_r\} \qquad \text{and} \qquad F_0 = \{q_1, \ldots, q_s\}.$$

Let $\alpha_{t+1} = 1$. Go to *Step* 7.

Step 12: Calculate the lower bound z_{t+1} for ϕ_{t+1} according to Lemma 8.5 and let

$$\pi := \pi \cup \{t + 1\}, \qquad t := t + 1 .$$

Go to *Step* 6.

Note 1. As Problem (P') is defined on a bounded polyhedron, if it has a feasible solution it also has an optimal solution. According to the theorem of duality so does Problem (D').

Note 2. If ϕ_0^0 turns out to be feasible for Problem (P) in Step 6, then it is also optimal. In other words the first feasible solution is usually optimal. Though it may seem to be advantageous, it is not always so. First of all, before the end of the calculation no feasible solution is available. If the computer run is terminated before the end, then we remain empty-handed. Secondly, the full branch-and-bound search may require a substantial core size. This may be reduced by using single-branch search if the tree is growing very large. For further details of branch-and-bound algorithms, see Chapter 3.

Note 3. The order of variables is fixed throughout the algorithm. This may be inconvenient. In particular, if ϕ_0^0 already satisfies filter (s-constraint) (8.9) the procedure must be changed. It may be altered in such a way that from time to time or if some conditions are met, constraint (8.9) is recalculated. Obviously in this case, all the generated new constraints must be saved and changed according to the last jump in the tree. This idea should be linked to the partial single-branch enumeration – for example in such a way that whenever a new filter is generated, the corresponding part of the tree is enumerated first.

Note 4. The tests and s-constraint constructions of Chapter 5 may also be applied.

Note 5. The problem discussed in Note 3 may be handled in another way. If ϕ_k already satisfies the filter, then $z(\phi_k)$ is no longer a lower bound for the unformed "brothers" (to the descendants of its immediate ascendant). In this case, if no new filter is generated then either the remaining descendants are also generated or another lower bound is used. For example, the least coefficient of the free variables is added to the lower bound of the ascendant. The methods of Note 3

and Note 5 may also be used alternately, depending on the size of the remaining tree.

Now the convergence of the algorithm is stated.

Theorem 8.2: *The above algorithm comes to an end after a finite number of steps. If Problem (P) has a feasible solution, then the optimal solution is also provided.*

The proof is decomposed into several lemmata:

Lemma 8.6: *Let* $\mathbf{x}^*$ *be an optimal solution of Problem (P). Then any time when Step 6 is executed*

$$z(\phi_{i_*}) = \min_{i \in \pi} z(\phi_i) \leqq \mathbf{c}^T \mathbf{x}^* . \tag{8.56}$$

Proof: First of all, let us suppose that in the algorithm an ordered pseudo-solutions occurs for which

$$\phi_k^0 = \mathbf{x}^* .$$

Then inequality (8.56) is satisfied because ϕ_k may be deleted only if ϕ_k is chosen sometimes in Step 6, but this is the end of the algorithm because then ϕ_k is a feasible solution. (Before deleting ϕ_k the above inequality must hold.)

Let us now suppose the opposite, i.e. no ϕ_k occurs in the algorithm such that

$$\mathbf{x}^* = \phi_k^0, \qquad k \in \pi. \tag{8.57}$$

But at the beginning

$$\mathbf{x}^* \in D^n = \{\phi_0\} \tag{8.58}$$

thus there exists a last ordered pseudo-solution such that

$$\mathbf{x}^* \in \phi_r, \qquad \mathbf{x}^* \neq \phi_r^0, \qquad r \in \pi$$

but no generated descendants of ϕ_k contain the vector $\mathbf{x}^*$. Now obviously

$$z(\phi_r) \leqq \mathbf{c}^T \mathbf{x}^*. \tag{8.59}$$

When subscript r is omitted from the set π, then either ϕ_r is a feasible solution – then the algorithm is ended – or one of its immediate descendants, say ϕ_s, is formed. According to the optimal indexing (if Note 3 is taken into account)

$$z(\phi_s) \leqq \mathbf{c}^T \mathbf{x}^*. \tag{8.60}$$

But the descendant of ϕ_r containing $\mathbf{x}^*$, say ϕ', is never formed according to our second supposition. It is possible to surpass the forming of ordered pseudo-solution ϕ' in Step 7 only if one of the foregoing descendants of ϕ_r, say ϕ_t, turns out to have a feasible root ϕ_t^0. Then ϕ_0 is an alternative optimum of the problem and the procedure is ended at the deletion of ϕ_t, which proves the lemma.

Lemma 8.7: *If*

$$\pi = \emptyset$$

when starting the execution of Step 6 any time in the algorithm, then Problem (P) has no solution.

Proof: This statement is an immediate consequence of the preceeding one. Namely if there exists an optimal solution of Problem (*P*), then the procedure may be ended only at Step 6 by recognizing an optimal solution. The mentioned lemma may be used formally if the convention $Z_0 = +\infty$ is introduced in the case $\pi = \emptyset$.

Lemma 8.8: *The above algorithm comes to an end in a finite number of steps.*

Proof: In the algorithm the period from one application of Step 6 to the next one is called a cycle. Obviously each cycle is finite as it consists of the decomposition of one pseudo-solution and some variable fixing in one pseudo-solution. Furthermore, the number of cycles must be finite because the application of Steps 6 and 7 is a decomposition for the deletion of a pseudo-solution, into subsets. Only a finite number of such decompositions is possible.

Proof of Theorem 8.2: The above three lemmata imply the theorem. Namely the algorithm is finished in a finite number of steps according to Lemma 8.8. This may happen at Steps 1, 2, or 6. In the first two cases, it is obvious that there is, respectively no solution and an optimal solution. If $\pi = \emptyset$ at the calculation of the minimal lower bound in Step 6, then according to Lemma 8.7 there is no solution. Finally, if ϕ_0^0 turns out to be a feasible solution, then it is also optimal because

$$z(\phi_0^0) = \min_{k \in \pi} z(\phi_k)$$

is always a lower bound of the optimum of Problem (*P*).

8.7 A numerical example

Let us solve the following problem obtained from the 2nd problem of Balas (1967) by omitting a few variables

$$\begin{aligned}
\min\ & 3x_1 + 4x_2 + 5x_3 + 6x_4 + 5x_5 + 3x_6 + 9x_7 + 4x_8 \\
& 6x_1 + 8x_2 - 9x_3 - 3x_4 + 8x_5 - 6x_6 + 2x_7 + 7x_8 \geqq 21 \\
& x_1 + x_2 + 6x_3 \qquad + 5x_5 + 9x_6 + 7x_7 - 2x_8 \geqq 10 \\
& -3x_1 - 6x_2 - 3x_3 + 9x_4 + 6x_5 + 7x_6 + 7x_7 + 7x_8 \geqq 14 \\
& x_j \in \{0, 1\} \qquad (j = 1, \ldots, 8).
\end{aligned}$$

Following the steps of the algorithm we obtain

Step 1: The optimal solution of Problem (P') is

$$x_3 = 0.88, \quad x_4 = 0.19, \quad x_5 = 1, \quad x_6 = 0.26, \quad x_8 = 1$$

$$x_1 = x_2 = x_7 = 0.$$

Step 2: There are fractional components.

Step 3: The optimal multipliers are

$$\hat{\mathbf{u}}^T = (0.78,\ 0.13,\ 0.93).$$

Problem $F(\hat{\mathbf{u}})$ may be written as

$$\min 3x_1 + 4x_2 + 5x_3 + 6x_4 + 5x_5 + 3x_6 + 9x_7 + 4x_8$$

$$2x_1 + 0.8x_2 + 5x_3 + 6x_4 + 12.4x_5 + 3x_6 + 8x_7 + 11.7x_8 \geqq 0$$

$$x_j \in \{0, 1\} \qquad (j = 1, \ldots, 8).$$

Step 4: Optimal indexing

Old subscript	1	2	3	4	5	6	7	8
New subscript	7	8	3	4	2	5	6	1

The problem is written down again for convenience introducing the changes in the variables.

$$\min 4x_1 + 5x_2 + 5x_3 + 6x_4 + 3x_5 + 9x_6 + 3x_7 + 4x_8$$

$$7x_1 + 8x_2 + 9x_3 - 3x_4 - 6x_5 + 2x_6 + 6x_7 + 8x_8 \geqq 21$$

$$-2x_1 + 5x_2 + 6x_3 + 9x_5 + 7x_6 + x_7 + x_8 \geqq 10$$

$$7x_1 + 6x_2 - 3x_3 + 9x_4 + 7x_5 + 7x_6 - 3x_7 - 6x_8 \geqq 14$$

$$x_j \in \{0, 1\} \qquad (j = 1, \ldots, 8),$$

and the filter is the following

$$11.7x_1 + 12.4x_2 + 5x_3 + 6x_4 + 3x_5 + 8x_6 + 2x_7 + 0.8x_8 \geqq 30.7.$$

Step 5: $\pi = \emptyset$, $t = 0$, $\phi_0 = (\emptyset)$, $\alpha_0 = 1$

Step 8; No result from tests $F2$ and $P2$.

Step 9: $F_1 \cup F_0 = \emptyset$

Step 10: $\phi_1 = (1)$, $\alpha_1 = 0$

TABLE 8.1

t	ϕ_t	z_t	π
1	1	15, 6	1
2	$\bar{1}$ 2	23, 8	1, 2
3	1 2	15, 6	2, 3
4	1 $\bar{2}$ 3	23, 6	2, 3, 4
5	1 2 3	15, 6	2, 4, 5
6	1 2 $\bar{3}$ 4	15	2, 4, 5, 6
7	1 2 3 4	20	2, 4, 6, 7
8	1 2 3 $\bar{4}$ 5	17	2, 4, 6, 7, 8
9	1 2 3 $\bar{4}$ $\bar{5}$ 6	23	2, 4, 6, 7, 8, 9
10	1 2 3 $\bar{4}$ $\bar{5}$ $\bar{6}$ 7	17	2, 4, 6, 7, 8, 9, 10
	1 2 3 $\bar{4}$ $\bar{5}$ $\bar{6}$ $\bar{7}$ 8		
11	1 2 $\bar{3}$ $\bar{4}$ 5	21	2, 4, 7, 8, 9, 10, 11
	1 2 $\bar{3}$ $\bar{4}$ $\bar{5}$ 6		
12	1 2 $\bar{3}$ 4 5	18	2, 4, 7, 8, 9, 10, 11, 12
13	1 2 $\bar{3}$ 4 $\bar{5}$ 6	23	2, 4, 7, 8, 9, 10, 11, 12, 13
14	1 2 $\bar{3}$ 4 $\bar{5}$ $\bar{6}$ 7	18	2, 4, 7, 8, 9, 10, 11, 12, 13, 14
15	1 2 $\bar{3}$ 4 $\bar{5}$ $\bar{6}$ $\bar{7}$ 8	19	2, 4, 7, 8, 9, 10, 11, 12, 13, 14, 15
16	1 2 3 $\bar{4}$ 5 6	26	2, 4, 7, 9, 10, 11, 12, 13, 14, 15, 16
17	1 2 3 $\bar{4}$ 5 $\bar{6}$ 7	20	2, 4, 7, 9, 10, 11, 12, 13, 14, 15, 16, 17
18	1 2 3 $\bar{4}$ 5 $\bar{6}$ $\bar{7}$ 8	21	2, 4, 7, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18
19	1 2 $\bar{3}$ 4 5 6	27	2, 4, 7, 9, 11, 13, 14, 15, 16, 17, 18, 19
20	1 2 $\bar{3}$ 4 5 $\bar{6}$ 7	21	2, 4, 7, 9, 11, 13, 14, 15, 16, 17, 18, 19, 20
21	1 2 $\bar{3}$ 4 5 $\bar{6}$ $\bar{7}$ 8	22	2, 4, 7, 9, 11, 13, 14, 15, 16, 17, 18, 19, 20, 21
22	1 2 3 4 5	23	2, 4, 9, 11, 13, 16, 17, 18, 19, 20, 21, 22
23	1 2 3 4 $\bar{5}$ 6	29	2, 4, 9, 11, 13, 16, 17, 18, 19, 20, 21, 22, 23
24	1 2 3 4 $\bar{5}$ $\bar{6}$ 7	23	2, 4, 9, 11, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24
25	1 2 3 4 $\bar{5}$ $\bar{6}$ $\bar{7}$ 8	24	2, 4, 9, 11, 13, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25

Optimal solution $\phi_{17} = (1, 2, 3, \bar{4}, 5, \bar{6}, 7, \bar{8})$
Rearranging the variables $x^* = (1, 0, 1, 0, 1, 1, 0, 1)$

Step 12: $z_1 = z_1(\phi_1) = 4 + 5 + \frac{6}{6} \times 1.6 = 15.6$

$\pi = \{1\}, \quad t = 1$

Step 6: $z_0 = 15.6 \ \phi_0 = \phi_1, \ \alpha_0 = \alpha_1 = 0, \ \pi = \emptyset$

ϕ_0^0 is not a feasible solution.

Step 7: $\phi_2 = (1, 2)$

$z_2 = 0 + 5 + 5 + 6 + 3 + \frac{9}{8} \times 4.3 = 23.8 \qquad \alpha_2 = 1$

$\pi = \{2\}, \quad t = 2$

Step 8: No results

Step 9: No results

Step 10: $\phi_3 = (1, 2), \quad \alpha_3 = 0$

Step 12: $z_3 = 15.6 \quad \pi = \{2, 3\}, \quad t = 3$

Step 6: $z_0 = 15.6 \quad \phi_0 = \phi_3, \quad \alpha_0 = \alpha_3 = 0, \quad \pi = \{2\}$

Step 7: $\phi_4 = (1, 2, 3) \quad \alpha_4 = 1$

$\pi = \{2, 4\} \quad t = 4.$

The algorithm can be continued in a similar way. The process of solution may be followed in Table 8.1. Instead of generating new filters, if the filter was satisfied by ϕ_k its next "brother", i.e. the next descendant of its immediate ascendant was also generated.

The new ordered pseudo-solution, the corresponding lower bound, and the actual covering system are denoted at each iteration. After forming pseudo-solution ϕ_{25}

$$\min_{i \in \pi} z(\phi_i) = z(\phi_{17}).$$

As ϕ_{17}^0 is a feasible solution of Problem (P) it is also optimal. Using the subscript tableau of Step 4 the result is retransformed to the original variable subscripts

$$\mathbf{x}^* = (1, 0, 1, 0, 1, 1, 0, 1).$$

CHAPTER 9

HEURISTICS IN DISCRETE PROGRAMMING

9.1 Introduction

Heuristic methods play a very important role in solving different kinds of integer programming problems. First, a great variety of exact algorithms may be accelerated by finding a good initial feasible solution and perhaps further ones throughout the algorithm. Secondly, good lower bound calculations for the objective function may be associated with most of the heuristic procedures (in the case of minimization problems; for maximization, they are substituted by upper bounds). These lower bounds may be used not only in branch-and-bound procedures but also in other methods for estimating the distance between the optimum and the best objective function value obtained so far. And finally, the capability of different exact algorithms is far beyond the need, especially in the number of discrete variables. For these cases, good heuristic algorithms may be used preferably with some estimations of the optimum.

Besides the above mentioned separate functions, if heuristics is considered in a broader sense, then we can say that most of the discrete programming algorithms are blended with heuristics. This includes tests for obligatory variable fixings and for showing the emptiness of pseudo-solutions, evaluations of variables and constraints, choice of variable sets for obligatory or suggested fixings, ordering and reordering of variables, calculations of lower and upper bounds and penalties, etc. And last but not least, the organization of more complicated or combined exact algorithms is also subject to heuristics. This will be discussed in the present chapter. As an example, a new algorithm for the set covering problem will be given in the next chapter. This algorithm consists of an exact branch-and-bound procedure which has a heuristic method built in. If certain conditions are met, the heuristic algorithm itself generates an optimal solution. In other cases it usually provides a very good lower bound of the optimum. This is a good example for the symbiosis of exact and heuristic methods.

9.2 Local search techniques

Let us consider an arbitrary finite set P, and the following discrete programming problem

$$\min \{f(p) \mid p \in P\}, \tag{9.1}$$

where the function $f(p)$ is defined on the set P. A great variety of heuristic procedures may be described by the neighbourhood structure and successor structure introduced by Reiter and Sherman (1965). The description of the method follows.

Define a neighbourhood for each element of the set P, which is a subset of P. Choose any element $p_1 \in P$ and search the above defined neighbourhood of point p_1 only. This search may include total enumeration of the neighbourhood or it may be continued in a predetermined order until no improvement in the objective function $f(p)$ is obtained. The best point obtained in the local search is denoted by p_2. If $f(p_2) < f(p_1)$, that is, $p_2 \neq p_1$, then the neighbourhood of point p_2 is considered for a local search. It is continued until $f(p_1) > f(p_2) > \ldots > f(p_{r-1}) = f(p_r)$, $p_{r-1} = p_r$. Then another element of set P is chosen as a starting point p_1. The random choice of new points is repeated until either we are reasonably sure that no improvement can be obtained in such a way, or the cost of further calculations is not justified by the expected decrease in the objective function.

In many cases, however, the domain of optimization is not given explicitly but is determined by inequalities, equations, logical constraints, etc. Therefore the more general problem

$$\min \{f(p) \mid p \in T \subset P\} \tag{9.2}$$

will be considered. The procedure of local search techniques remains esentially the same. The only difference is that no objective function improvement is accepted if the constraints are more violated. (A precise definition is introduced below.)

Usually only the implicitly determined finite discrete set T is given and it must be embedded in another discrete set P, the elements of which may easily be generated. If the discrete set T contains a countably infinite number of elements, e.g. the vectors of nonnegative integer components in the n-dimensional Euclidean space, then either explicit upper bounds of each component must be determined from the constraints or, alternatively, other methods described in the following sections should be used.

Let us now generalize the definition of Reiter and Sherman for problem (9.2).

Definition 9.1: *A successor structure for problem (9.2) is defined by a triple* (N, μ, s), *where*

(i) N *is a collection of subsets* $\{N(p) \mid p \in P\}$, *such that*

  (a) *for each* $p \in P$ *there is exactly one* $N(p) \in N$

  (b) $p \in N(p)$ *for each* $p \in P$

(ii) μ *is a measure of the infeasibility of set* P, *for which*

  (a) $\mu(p) \geqq 0$ *for each* $p \in P$

  (b) $\mu(p) = 0$ *if and only if* $p \in T$

(iii) s *is a successor function defined on* P *with the properties*

  (a) $s(p) \in N(p)$

  (b) either $\mu(s(p)) < \mu(p)$ or $\mu(s(p)) = \mu(p)$ and $f(s(p)) \leqq f(p)$,

  (c) if $\mu(s(p)) = \mu(p)$ and $f(s(p)) = f(p)$, then $s(p) = p$,

(d) if $s(p) = p$, *then* $\mu(p') \geqq \mu(p)$ *for each* $p' \in N(p)$, *furthermore* $f(p'') \geqq f(p)$ *for each* $p'' \in N(p)$, *with* $\mu(p'') = \mu(p)$.

$N(p)$ is called the neighbourhood of point p. The infeasibility measure is usually defined in such a way that each constraint is checked for a point $p \in P$. If a constraint is satisfied, then its contribution to the measure value $\mu(p)$ is zero. In the opposite case, a value is calculated which expresses how far we are from satisfying the constraint. The individual infeasibility values are added up in $\mu(p)$. Sometimes weights are also used for the contributions obtained from the differens constraints. These weights are proportional to the difficulty of the constraints and they may be calculated, e.g. by some learning procedures. The successor function is defined in such a way that it seeks better feasiblity first. To obtain the largest improvement both in the feasibility and in the objective function value, the entire neighbourhood $N(x)$ is fathomed. Usually complete enumeration is used for this purpose but more sophisticated methods can also be tried.

If $T = P$, then $\mu(p) \equiv 0$ and the original definition of Reiter and Sherman results.

Example 1: Reiter and Sherman discussed heuristic algorithms for the travelling salesman problem, which was formulated in Chapter 1 and for which a branch-and-bound solution method was described in Chapter 3. The most successful heuristic algorithm (detailed in the cited paper), the Algo IV(r) series, may be summarized in the following way.

Let P be the set of all permutations of the integers $1, 2, \ldots, n$ and let f be a real valued function defined on P. The function f is to be minimized. Because set P is explicitly given, $T = P$ and $\mu(\mathbf{p}) \equiv 0$. Let $\mathbf{p} = (j_1, j_2, \ldots, j_n)$ and $1 \leqq r < n$ arbitrary integer. Then

$$N_r(\mathbf{p}) = \{(j_1, \ldots, j_r, j_{r+1}, \ldots, j_n), (j_{r+1}, j_1, \ldots, j_r, j_{r+2}, \ldots, j_n),$$

$$(j_{r+1}, j_{r+2}, j_1, \ldots, j_r, j_{r+3}, \ldots, j_n), \ldots, (j_{r+1}, \ldots, j_n, j_1, \ldots, j_r)\}$$

defines the neighbourhood of point $\mathbf{p}$ in Algo IV(r). Let $\mathbf{p}'$ be the first element for which $f(\mathbf{p}') < f(\mathbf{p})$ then

$$s(\mathbf{p}) = \mathbf{p}'.$$

If no such $\mathbf{p}'$ exists, then

$$s(\mathbf{p}) = \mathbf{p}.$$

The elements of the Algo IV(r) series are applied consecutively. This means that Algo IV(1) is tried first, i.e. only one element of $\mathbf{p}$ is shifted, according to $N_1(\mathbf{p})$. Then two elements are shifted together in Algo IV(2) as shown in $N_2(\mathbf{p})$ etc. Whenever a better solution is obtained, Algo IV(1) is started again followed by IV(2) etc. Very good results are cited by the authors for up to 57-city problems. For each problem, at least as good solutions were obtained as the best one avail-

able earlier, with the computer time remaining resonably small. Estimations of the computing time and expected gain are compared and used to stop the generation of new starting points.

Example 2: A similar approach may be used for the so called restricted assignment problem which is described and solved by a heuristic method in Kovács and Nagykálnai (1968). The method used in this paper is of Lagrangian multiplier type and an estimation of the error is also given, furthermore an agricultural application is presented together with the analysis of a small-sized real life problem. The mathematical problem may be described as follows:

$$\begin{aligned} &\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\ &\sum_{i=1}^{n} x_{ij} = 1 \quad (j = 1, \ldots, n) \\ &\sum_{j=1}^{n} x_{ij} = 1 \quad (i = 1, \ldots, n) \\ &\sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij} \leqq K \\ &x_{ij} \in \{0,1\} \quad (i, j = 1, \ldots, n) . \end{aligned} \tag{9.3}$$

This is almost the ordinary assignment problem which means a one-to-one assignment of n machines and n jobs at a minimal total cost. The cost of assigning machine i to job j is c_{ij}. The only difference is in the inequality which imposes a further restriction. This is used if another source (e.g. manpower) is also taken into account. For example, if machine i is assigned to job j, then a manpower demand d_{ij} arises. The upper bound K for the manpower is given. This restriction is very natural in agricultural applications, where different plant crops are assigned to different fields. In this case the required manpower largely depends on the assignment.

Feasible solutions to the assignment problem may easily be described by the permutations of the positive integers $(j_1, j_2, \ldots, j_n)$ as in Example 1. There is a big difference however in the corresponding solution. In the travelling salesman problem this represents the solution

$$x_{j_1 j_2} = x_{j_2 j_3} = \ldots = x_{j_{n-1} j_n} = x_{j_n j_1} = 1$$

and in the assignment problem

$$x_{1 j_1} = x_{2 j_2} = \ldots = x_{n j_n} = 1 .$$

Obviously not all permutations correspond to a solution of the resticted assignment of elements $1, 2, \ldots, n$, but the set

$$T = \{\mathbf{p} \mid \mathbf{p} \in P, \quad d_{1 j_1} + d_{2 j_2} + \ldots + d_{n j_n} \leqq K\}$$

is also introduced, where P is the set of permutations of the first n positive integers, that is,

$$\mathbf{p} = (j_1, j_2, \ldots, j_n).$$

Now the infeasiblity measure

$$\mu(\mathbf{p}) = \left(\sum_i d_{ij_i} - K\right)^+$$

is also introduced where the superscript + means "positive part".

Now the neighbourhoods $N_r(\mathbf{p})$ introduced in example 1 may be used without any change. Moving from one element of this neighbourhood to another however has an entirely different meaning and corresponds to a different calculation in the cost matrix. Furthermore, in (9.3) not all points of set P are permitted, thus feasible solutions are searched first. This means that the successor function should be modified. Let us consider any point $\mathbf{p} \in P$. Let $\mathbf{p}'$ be the first element in the presently considered neighbourhood $N_r(\mathbf{p})$, for which either

$$\mu(\mathbf{p}') < \mu(\mathbf{p})$$

or

$$\mu(\mathbf{p}') = \mu(\mathbf{p}) \qquad \text{and} \qquad f(\mathbf{p}') < f(\mathbf{p}),$$

where

$$f(\mathbf{p}) = c_{1j_1} + c_{2j_2} + \ldots + c_{nj_n}.$$

Let $s(\mathbf{p}) = \mathbf{p}'$ and if no such $\mathbf{p}'$ exists, then $s(\mathbf{p}) = \mathbf{p}$. It makes sense to use the following alternative definition for $\mathbf{p}'$: *It is the first element in* $N_r(\mathbf{p})$ *for which*

$$\mu(\mathbf{p}') < \mu(\mathbf{p}),$$

or if $\mu(\tilde{\mathbf{p}}) \geqq \mu(\mathbf{p})$ *for all* $\tilde{\mathbf{p}} \in N_r(\mathbf{p})$, *then*

$$f(\mathbf{p}') + \lambda\mu(\mathbf{p}') < f(\mathbf{p}) + \lambda\mu(\mathbf{p}),$$

where $\lambda \geqq 0$ is a constant or a parameter of the procedure. (See Note 3 at the end of this section.)

In this second definition the objective function is considered only if either a feasible solution is found or if no improvement in feasibility can be achieved in the neighbourhood.

Example 3: Let us consider the following problem

$$\min \{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} \in T\}, \tag{9.4}$$

where

$$T = \{\mathbf{x} \mid \mathbf{A}\mathbf{x} \geqq \mathbf{b},\ x_j \in \{0, 1\} \quad (j = 1, \ldots, n)\},$$

$\mathbf{A}$ is a given matrix of size $m \times n$, $\mathbf{b}$ and $\mathbf{c}$ are given vectors of corresponding size. The elements of set T are not easily determined, therefore the set

$$P = \{\mathbf{x} \mid x_j \in \{0, 1\} \quad (j = 1, \ldots, n)\}$$

is also introduced. The r neighbourhoods of vectors $\mathbf{p} \in P$ are defined by permitting at most r components to be changed

$$N_r(\mathbf{p}) = \left\{ \mathbf{x} \mid \mathbf{x} \in P, \quad \sum_{j=1}^{n} |x_j - p_j| \leqq r \right\},$$

where $1 \leqq r \leqq n$ is an arbitrary integer, but usually $r = 1, 2$ or 3.

The infeasibility measure is nothing but the sum of quantities by which the constraints are violated

$$\mu(\mathbf{p}) = \sum_{i=1}^{m} (b_i - \mathbf{a}^i\mathbf{p})^+,$$

where a^i is row i of matrix $\mathbf{A}$ and the superscript $+$ again stands for "positive part". If further information is available on the question regarding how difficult the constraints are, then the modified form

$$\mu(\mathbf{p}) = \sum_{i=1}^{m} w_i(b_i - \mathbf{a}^i\mathbf{p})^+$$

can be used, where $w_i \geqq 0$ is the weight of constraint i. These weights may be obtained by any of the methods described in the present and the next chapters.

The successor function may be defined as in the foregoing examples (the first better point in $N_1(\mathbf{p})$; if no such point exists, then the first better point in $N_2(\mathbf{p})$ etc.). Better point here means that as a first criterion better in feasibility; or as a second criterion, better in objective function.

In choosing the largest r for which the neighbourhoods $N_r(\mathbf{p})$ are considered, it should be taken into account that the number of elements in $N_r(\mathbf{p})$ is

$$1 + \binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{r}.$$

The order of variables may be chosen arbitrarily in each step according to prior knowledge or experience with the problem.

Example 4: As a last example, let us consider a problem of design assortment for a ready-made clothing-factory.

Notation

m	number of rolls
s	number of different sizes ordered
t	number of patterns
L	maximum number of pieces that can be cut together
h_i	length of roll i
d_j	demand for cloth size j

e_k	length of pattern k
a_{kj}	the number of clothes of size j in pattern k
v	number of different patterns permitted for one roll
x_{ik}	the number of pieces of pattern k to be cut out from roll i
$g(u)$	the value of material of length u

Function $g(u)$ is piece-wise linear and its value is 0 under a certain threshold u_0. Now the problem is to satisfy the demand d_j and obtain a maximal value for the remaining material. The rest of the constraints speak for themselves

$$\begin{aligned} &\max \sum_{i=1}^{m} g\left(h_i - \sum_{k=1}^{t} e_k x_{ik}\right) \\ &\sum_{k=1}^{t} e_k x_{ik} \leqq h_i \quad (i = 1, \ldots, m) \\ &\sum_{i=1}^{m} \sum_{k=1}^{t} a_{kj} x_{ik} \geqq d_j \quad (j = 1, \ldots, s) \\ &\sum_{k=1}^{t} \left\langle \frac{x_{ik}}{L} \right\rangle \leqq v \quad (i = 1, \ldots, m) \\ &x_{ik} \in \{0, 1, 2, \ldots\} \quad (i = 1, \ldots, m;\ k = 1, \ldots, t). \end{aligned} \tag{9.5}$$

The sign $\langle d \rangle$ means the least integer not smaller than d. An alternative problem may be formulated in a similar way if the demand can be oversatisfied arbitrarily and profitably; it is necessary that the ratios of the different sizes be kept between given bounds.

Now set T is the set of $\mathbf{x}$ vectors satisfying the constraints of (9.5) and

$$P = \{\mathbf{x} \mid x_{ik} = 0, 1, \ldots, u_{ik} \quad (i = 1, \ldots, m;\ k = 1, \ldots, t)\},$$

where

$$u_{ik} = \left\lceil \frac{h_i}{e_k} \right\rceil.$$

The neighbourhoods could be defined as in the foregoing examples: the variables may be changed by at most r units

$$N_r(\mathbf{p}) = \{\mathbf{x} \mid \mathbf{x} \in P, \ \sum_{i,k} |x_{ik} - p_{ik}| \leqq r\}.$$

On the other hand, this choice of neighbouring points does not fit the special structure of the problem. There are a great number of possibilities. We shall consider two of them to illustrate the ideas. First of all let us suppose that the demands are not yet met: for example d_1 pieces of size 1 are not yet available. Then the following neighbourhoods can be defined

$$N_r(\mathbf{p}) = \{\mathbf{x} \mid \mathbf{x} \in P, \ \sum_{i} \sum_{k \in J} (x_{ik} - p_{ik}) \geqq r\} \cup \{\mathbf{p}\}$$

where $$J = \{k \mid a_{k1} > 0\}.$$

The interesting thing in this neighbourhood definition is that only certain components are changed by the search process, the others may be changed according to the other requirements, the length of rolls, etc.

The second kind of neighbourhood is illustrated in the case when solutions satisfying the demands are to be improved according to the objective function. Then the satisfaction of the demands may be retained, for example, by fixing the number of patterns used, i.e. only changes between different rolls are permitted.

$$N_r(\mathbf{p}) = \{\mathbf{x} \mid \mathbf{x} \in P, \quad \sum_i x_{ik} = \sum_i p_{ik} \quad (k = 1, \ldots, t); \quad \sum_i \sum_k |x_{ik} - p_{ik}| \leqq r\}.$$

The infeasibility measure may be defined as before

$$\mu(\mathbf{p}) = \lambda \sum_{i=1}^{m} \left(\sum_{k=1}^{t} e_k x_{ik} - h_i \right)^+ + \sum_{j=1}^{s} \left(d_j - \sum_{i=1}^{m} \sum_{k=1}^{t} a_{kj} x_{ik} \right)^+ .$$

The last group of inequalities may be kept by introducing not too many different patterns for each roll. The multiplier λ helps to compose the two groups of constraints: the demands, and the roll length.

If we use the special neighbourhoods suggested for this and similar problems, then full search of the neighbourhoods is not advisable, and is not even necessary. The constraints may be used to decrease the number of elements to be examined. For example the roll length restricts the search. Further restrictions may be implied by the objective function if we avoid too many short remnants, etc.

Note 1: As a conclusion of the above examples the neighbourhood method or local search technique is not a readily available and applicable heuristic method, but rather an idea which may be used for a wide variety of discrete programming problems. In each case it is applied, the problem should be thoroughly studied and special neighbourhoods, infeasibility measures, and successor functions have to be used.

Note 2: The main disadvantage of the local search techniques is that no estimation of the optimum is available. As a means of increasing the probability of finding the real optimum, several starting points are to be chosen which are spread over the entire set P. For this purpose either a thorough deterministic design is used or a probability distribution is defined on the set P. The later choices of starting points may depend on the earlier result, e.g. the districts which turn out to be better may be chosen with a higher probability. The number of starting points can be recalculated after each iteration (one local search) on the basis of the computing cost and expected return in terms of the objective function decrease. Local search techniques can be combined with other heuristic and exact methods providing lower bounds of the objective function.

Note 3: Sometimes temporarily worse points are also permitted as next steps (worse in the sense of the feasibility measure or in the objective function value). The reason for this is illustrated by the following simple numerical example

$$10x_1 - 8x_2 \geqq 1$$

$$-7x_1 + 9x_2 \geqq 2.$$

If only neighbourhoods $N_1(\mathbf{p})$ are used, then $\mathbf{p} = (0, 0)$ is a local optimum, because

$$\mu(0, 0) = 3, \qquad \mu(1, 0) = 9, \qquad \mu(0, 1) = 9.$$

But on the other hand, only one point further $\mu(1,1) = 0$. Naturally in $N_2(\mathbf{p})$ the problem disappears, but similar examples may be shown for any r neighbourhoods. It can happen that the higher rank neighbourhoods, which would solve the trouble require a great amount of computation. But by breaking the rule of monotonic improvement of infeasibility (or objective function value) just for a limited number of steps, much better solutions may be found. In this case finiteness of the procedure should be ensured and new guides must be found instead of the improved value of infeasibility measure. Such a guide might be, e.g. one of the constraints alone that is difficult to satisfy.

9.3 Relaxation methods

The common property of all these methods is that the constraints are relaxed in some way. Either some of the functional and integrality constraints are deleted or they are substituted by a weaker system of constraints in such a way that no feasible solution of the original problem is deleted. Furthermore the relaxed problem must be easy to solve. If the optimal solution of the relaxed problem is feasible for the original problem, then it is also optimal and the problem is solved. In the opposite case no better solution may be obtained in the more restricted original problem, i.e. a lower bound has been obtained for the optimum (in the case of minimization problems). Therefore the relaxation methods provide an approach opposite to the local search techniques. The latter usually result in good feasible solutions and thus determine upper bounds of the optimum of minimization problem. If both approaches are used, then good estimations of the real optimum are always available and this may guide both the search and the generation of new relaxed problems. They may even be built into exact algorithms. The most obvious choice for this purpose is the branch-and-bound principle because it presupposes the knowledge of good bounds. But most of the other methods can also be made more effective by using either the relaxation methods or the search techniques or both. We shall now review the most common relaxation methods individually.

(a) Surrogate constraints

Surrogate constraints can be obtained from the inequalities of the original problem by forming their nonnegative linear combinations. These were discussed in detail in Chapter 5 and are mentioned here only for completeness.

(b) Continuous version

The continuous version of a discrete programming problem may be considered at any time, omitting the integrality requirements or substituting them by lower and upper bounds of the variables. This is done in all cutting methods. Their substance is to solve the continuous problem and check if the integrality requirements are satisfied by chance. In the case of a positive answer the discrete problem is also solved. In the opposite case a new restriction is introduced which excludes the, at present, optimal but noninteger optimal solution, but it does not exclude any of the feasible lattice points. The new continuous problem is solved again. The first cutting plane algorithm was found by Gomory (1958). A short discussion is to be found Chapter 5, a detailed description of the method is in Gomory (1963). This is in fact an exact method but it can be used as a heuristic tool for accelerating other algorithms. Several other cuts have since been designed; a promising one was found by Balas (1971).

The continuous version of a discrete programming problem is frequently use both for pure and for mixed cases if a branch-and-bound method is applied. More details are available in Chapter 3.

(c) The leaving out of nonnegativity restrictions

The leaving out of nonnegativity restrictions is an opposite approach to the one described in point *b*. The continuous optimum is calculated first, as before. Then basic variables of the optimal solution are expressed by the nonbasic variables. Following this, the integrality of the basic variables is taken into account with the help of a congruence system, but their nonnegativity restriction is temporarily deleted. If the optimal solution of this new problem satisfies the nonnegativity requirements, then the optimal solution of the original problem is available. In the opposite case a further search is induced in the order of monotone nondecreasing objective function values. Both the heuristic part and the complete exact algorithm are described in Chapter 4.

(d) Leaving out some variables

The leaving out of some variables is again a possibility for further heuristics. On the basis of prior knowledge or experience obtained in the solution process some of the variables can be expected to be zero. Some evaluations may also show

that their use in positive level is unfavourable from the point of view of direct and indirect consequences. A direct consequence is the increase in the infeasibility value and/or in the objective function value. Indirect consequences are the ones induced by the use of these variables: obligatory fixings and consequences which may be obtained by analysing the individual requirement infeasibilities. In the case of 0–1 problems similar arbitrary variable fixings may be used at value 1 if they are likely to be in the optimal solution. Some further details are given on the evaluations of variables in Chapter 11.

(e) Optimization over the cone

Let us solve again the continuous version of a linear discrete programming problem. For simplicity, let us now consider the optimal basic solution which is supposed as being unique. This corresponds to a vertex of the convex polyhedron defining the feasibility region of the continuous problem. The active constraints, i.e. the hyperplanes containing the optimal solution, form a polyhedral cone. This cone, as it contains the polyhedron, can be considered as the relaxation of the original convex polyhedron. Thus a search procedure can be used for determining the best lattice point of this cone. If it is not yet feasible for the original constraints, then the next best lattice point is determined, etc. The first feasible lattice point will also be optimal. Many heuristic and exact algorithms are based on this idea. Two of them will be discussed in the next points.

(f) Fourier–Motzkin elimination

The Fourier–Motzkin elimination aims at reducing the number of variables in any linear programming problem – as it is described, for example, in Dantzig (1963). The only difficulty is that the price for the smaller number of variables is usually an enormous increase in the number of constraints. This is not the case if it is applied for the so-called Hermitian cone problem. The papers of Bradley (1971a), (1971b), and Bradley and Wahi (1973) show how the problem

$$\begin{aligned} &\max \mathbf{c}^T\mathbf{x} \\ &\mathbf{A}\mathbf{x} \leqq \mathbf{b} \\ &\mathbf{x} \geqq \mathbf{0} \\ &\mathbf{x} \text{ integer} \end{aligned} \tag{9.6}$$

can be transformed into the $n!$ equivalent problem of special form (the Hermitian Canonical form) by using unimodular transformations of type $\mathbf{x} = \mathbf{h} + \mathbf{K}\mathbf{w}$, where $\mathbf{h}$ is a fixed and $\mathbf{w}$ is a variable integer vector and $\mathbf{K}$ is unimodular. (The

subdeterminants of $\mathbf{K}$ are ± 1.) The cone described in point e is called in this case a Hermitian cone and the corresponding problem takes the following form

$$\begin{aligned} \max\ & g_1x_1 + g_2x_2 + \ldots + g_nx_n \\ & d_{11}x_1 \qquad\qquad\qquad\qquad \geqq f_1 \\ & d_{21}x_1 + d_{22}x_2 \qquad\qquad\qquad \geqq f_2 \\ & d_{n1}x_1 + d_{n2}x_2 + \ldots + d_{nn}x_n \geqq f_n \\ & x_j \text{ integer} \qquad (j = 1, \ldots, n) \end{aligned} \tag{9.7}$$

where $-d_{ii} < d_{ij} \leqq 0$ for all i, j with $j < i$ and $-d_{ii} < f_i \leqq 0$. Furthermore if (9.6) is bounded, then $g_j \leqq 0$ $(j = 1, \ldots, n)$. If the unknown objective function value is denoted by z, then the objective function of (9.7) is substituted by the inequality

$$g_1x_1 + g_2x_2 + \ldots + g_nx_n \geqq z\,.$$

For the inequality system of $n + 1$ inequalities so obtained, the Fourier–Motzkin elimination may be applied without increasing the number of constraints, thus the cone problem can easily be solved.†

(g) Set covering problems

The set covering problems may also be considered as a relaxation of a general linear integer problem. For simplicity let us consider the following problem:

$$\begin{aligned} & \min \mathbf{c}^T\mathbf{x} \\ & \mathbf{Ax} \geqq \mathbf{b} \\ & x_j \in \{0, 1\} \qquad (j = 1, \ldots, n) \end{aligned} \tag{9.8}$$

where $\mathbf{A}$ is given arbitrary $m \times n$ matrix, $\mathbf{c} \geqq \mathbf{0}$ and $\mathbf{b}$ are given vectors of corresponding size. The set covering problem differs from (9.8) in the coefficients: $a_{ij} = 0$ or 1 and $b_i = 1$. The general set covering problem has right hand sides $b_i > 0$, integer. The set covering problem and an algorithm for obtaining its optimal solution will be discussed in detail in Chapter 10. Now only a relaxation

† The basic ideaf or reducing the number of variables (as described in the above-mentioned literature) is the following: One of the variables is expressed from each inequality containing it. (The variable must stand alone at one side of the inequality with coefficient $+1$.) Then the variable may be omitted if we require that all expressions $\leqq x_j$ (the variable being omitted) must be less than or equal to all expressions $\geqq x_j$. This method is applied to the Hermitian cone problem in a reverse order of variable subscripts. [See problem (9.7)]

of (9.8) is shown, which is a set covering problem. For this purpose it is sufficient to see how 0–1 coefficient consequences of a single row

$$d_1x_1 + d_2x_2 + \ldots + d_nx_n \geqq b_0 \tag{9.9}$$

can be formed. Let us suppose that inequality (9.9) is not trivially satisfied, that is, $b_0 > 0$. Furthermore, let us suppose that the coefficients are already ordered (the subscripts are chosen in such a way)

$$d_1 \geqq d_2 \geqq \ldots \geqq d_r > 0 \geqq d_{r+1} \geqq \ldots \geqq d_n .$$

Let us now choose subscript p such that

$$\sum_{j=p+1}^{r} d_j < b_0 \leqq \sum_{j=p}^{r} d_j . \tag{9.10}$$

Then obviously the constraint

$$x_1 + x_2 + \ldots + x_p \geqq 1 \tag{9.11}$$

is a consequence of inequality (9.9) Different inequalities may also be obtained for the general set covering problem. For this purpose determine the subscripts s and t such that $t < s$ and

$$\sum_{j=s+1}^{r} d_j < b_0 - \sum_{j=1}^{t} d_j \leqq \sum_{j=s}^{r} d_j . \tag{9.12}$$

In this case the constraint

$$x_1 + x_2 + \ldots + x_s \geqq t + 1 \tag{9.13}$$

is a consequence of inequality (9.9). Several constraint of this type can be obtained depending on the choice of t.

Obviously each constraint of (9.8) can be used for generating such 0–1 coefficient constraints and so can their nonnegative linear combinations. (See Chapter 5 for generating s-constraints.)

9.4 A tightening method

An opposite approach to relaxation would be the tightening of the constraints. At first, it is not clear what can be gained because usually the original restrictions are rather tight or they become tighter by introducing the stronger and stronger objective function constraint. In the method described below, the tightening is applied to the cone problem outlined in Section 9.3 point e, which is actually a relaxation of the original discrete problem. The primary purpose of tightening is to ensure that the lattice point, obtained by rounding the continuous optimal solution, is in the first cone. Furthermore, a parametric rounding procedure is

generated along the lines connecting the optimal solutions of the continuous cone problem and its restricted version. The heuristic method was presented by Hillier (1969–1) and further development is made to obtain an exact enumeration algorithm [Hillier (1969–2)]. It can be summarized for the following problem

$$\begin{aligned} \max x_0 &= \sum_{j=1}^{n} c_j x_j \\ \sum_{j=1}^{n} a_{ij} x_j &\leqq b_i \qquad (i = 1, 2, \ldots, m) \\ x_j &\geqq 0 \qquad (j = 1, 2, \ldots, n) \\ x_j &\text{ integer} \qquad (j = 1, 2, \ldots, n) \end{aligned} \tag{9.14}$$

by the following steps.

Step 1: Solve the continuous version of (9.14) and denote the optimal solution by $\mathbf{x}^{(1)}$, the optimal basis by $\mathbf{B}$, and the set of subscripts of the basic variables by J_B. Furthermore, let I denote the set of row indices satisfied by $\mathbf{x}^{(1)}$ as an equation, and

$$J = \{j \mid x_j^{(1)} = 0\}.$$

Step 2: Determine $\mathbf{x}^{(2)}$, the optimal solution of the problem

$$\begin{aligned} \max x_0 &= \sum_{j=1}^{n} c_j x_j \\ \sum_{j=1}^{n} a_{ij} x_j &\leqq b_i^{(2)} \qquad (i \in I) \\ x_j &\geqq 0.5 \qquad (j \in J), \end{aligned} \tag{9.15}$$

where

$$b_i^{(2)} = b_i - \frac{1}{2} \sum_{j \in J_B} |a_{ij}| \qquad (i \in I) \tag{9.16}$$

or alternatively

$$b_i^{(2)} = b_i - \sqrt{n}/2 \qquad (i \in I). \tag{9.17}$$

Step 3: Search the vectors

$$\mathbf{x}^3 = (1 - \alpha)\mathbf{x}^{(1)} + \alpha \mathbf{x}^{(2)} \qquad 0 \leqq \alpha \leqq 1$$

to obtain different lattice points by rounding $\mathbf{x}^3$. For each different lattice point apply also a local search, as described in Section 9.2. The purpose is to find as good feasible solutions as possible.

Note 1: To determined $\mathbf{x}^2$, only a matrix multiplication is required because the same basis inverse may be used.

Note 2: If the right hand sides (9.16) are chosen, then the rounding of $\mathbf{x}^{2)}$ is obviously in the first cone because

$$\sum_{j=1}^{n} a_{ij}[x_j^{(2)}] \leqq \sum_{j=1}^{n} a_{ij}x_j^{(2)} + \frac{1}{2}\sum_{j \in J_B} |a_{ij}| \leqq b_i^{(2)} + \frac{1}{2}\sum_{\in J_B} |a_{ij}| = b_i .$$

If right hand sides (9.17) are chosen, then again the rounded point $[\mathbf{x}^{(2)}]$ must be in the original cone as the distance between $\mathbf{x}^{(2)}$ and $[\mathbf{x}^{(2)}]$ cannot be larger than $\sqrt{n}/2$.

Note 3: In the search procedure the series $\alpha = \varepsilon, 2\varepsilon \ldots$ may be used with a sufficiently small positive ε. It is also possible that α is changed by such a quantity that at least one component of the rounded vector $[\mathbf{x}^3]$ be different.

Note 4: Each time a new feasible solution has been found, then the objective function constraint is tightened accordingly, to permit only better solutions than the best one obtained so far. The procedure can be modified since the feasible region is getting smaller and smaller. For example, in the search procedure a standpoint may be to remain as far as possible from the boundary of already satisfied constraints. (Further details are given in Section 9.2.)

9.5 Probabilistic methods

If probabilistic procedures are tried the most obvious method would be a kind of Monte Carlo method or statistical sampling. This possibility was mentioned in Section 9.2 but not emphasized so much as in the paper of Reiter and Sherman (1965). The main reason for this different point of view is the presence of restrictions in the problems considered which make the statistical sampling ineffective, because even feasible solutions are quite rarely found. In the present section, however, an entirely different approach will be discussed, this approach is based on the results of Graves and Whinston (1968).

Let us consider, for simplicity, the problem

$$\begin{aligned} &\min \sum_{j=1}^{n} c_j x_j \\ &\sum_{j=1}^{n} a_{ij}x_j \geqq b_i \qquad (i = 1, \ldots, m) \\ &x_j \in \{0, 1\} \qquad (j = 1, \ldots, n) . \end{aligned} \tag{9.18}$$

Let us suppose that an enumeration method is used for solving this problem (any of those described in Chapters 2, 5 and 6). The probabilistic approach is used when no obligatory fixings can be found and free choices of variables are per-

mitted. These free choices may however influence the remaining calculations to a great extent. The more likely we are to get a feasible solution, the better. Therefore the free variables can be evaluated according to the expected number of feasible solutions after fixing them at value 1. To simplify the discussion let us consider only a single constraint from problem (9.18).

$$\xi = \sum_{j=1}^{n} d_j x_j \geqq d_0, \quad x_j \in \{0, 1\} \qquad (j = 1, \ldots, n). \tag{9.19}$$

Now x_j can be considered as random variables taking the values 0 and 1 with probability 0.5. The random variable ξ is the left hand side of the constraint and thus

$$1 - P(\xi < d_0) \tag{9.20}$$

is the relative frequency with which a random vector $\mathbf{x}$ satisfies the constraint. This is nothing but the ratio of the number of feasible vectors (for constraint 9.19) divided by the number of all such vectors, 2^n. In other words probability (9.20) is what we need. The expected value and standard deviation of variable ξ are easily calculated

$$E(\xi) = \frac{1}{2} \sum_{j=1}^{n} d_j$$

$$\sigma^2(\xi) = \frac{1}{4} \sum_{j=1}^{n} d_j^2 .$$

But the probability distribution of random variable ξ may be approximated by the normal distribution of the same parameters if n is sufficiently large, according to the theorem of Lindeberg [see e.g. Feller (1966)].

Mathematical tables and computational methods are available for calculating the values of the normal distribution; thus, the evaluating numbers for the feasibility regions of a single inequality constraint can be calculated.

The method may be extended to several constraints of type (9.20) and the common distribution of similarly defined random variables $\xi_1, \ldots, \xi_m$ is an m-dimensional normal distribution the parameters of which may easily be calculated from the coefficients of the constraints.

Though the procedure is practical only if the number of rows $m = 1, 2$ or at most 3, it can be used for constraints of special interest, e.g. the s-constraints of Chapter 5 and the objective function constraint.

9.6 Learning procedures

Such procedure utilize the information obtained throughout the former steps of the algorithm on the particular problem being solved. One way of doing so is to use different counters for certain events. For example, if we wish to measure the difficulty of the different constraints, then each time a trial solution does not

satisfy a constraint the corresponding counter in increased by one. Similar counters can also be used for the variables. It should be noted however, that after fixing a few variables the picture may be completely changed. Thus, the use of counters can be effective only if a kind of forgetting procedure is built in. In other words, the information from the last few steps must have a larger weight.

Another way of learning is the constant measuring of efficiency of different alternative subroutines. This can be done by comparing the running time and the "result" of different subroutines. These "results" may be: obligatory fixings, the finding of new feasible solutions, the fathoming of big solution sets, etc. – depending on the function of the routine in question. If the overall performance of each routine is evaluated for several problems, then we are able to obtain experience as to which routines should be used and at what parameter values. But this is not a learning procedure. On the other hand, if the central organization of the algorithm decides – for different problems and different parts of the solution process – which routines should be used and what are their actual parameters, then learning procedures for gathering information on the structure of the problem are very useful.

The above ideas will be illustrated and explained in more detail in Chapter 11 by describing a complex system of procedures designed for solving larger discrete programming problems.

If Monte Carlo methods or other simulation techniques are used for generating starting points in local search procedures (see Section 9.2), then better and worse districts may be found by learning procedures. Only the results obtained at each starting point need be analysed. Then points from better districts are chosen with higher probability.

9.7 Testing and evaluating

Tests may be defined as sufficient conditions for obligatory variable fixings and for certain pseudo-solutions being empty. Several tests were discussed in Chapter 5 and further ones can easily be designed especially for problems with particular structures. To a certain extent they may also be considered as heuristic tools because their efficiency and effect on the main algorithm is hardly predictable.

As was already mentioned in the foregoing sections, the evaluation of variables is usually very important when a free choice is to be made. One possibility is the use of different feasibility measures (see Section 9.2) applied after fixing the variables in question individually and temporarily. Another one was mentioned in Section 9.6. Naturally, the consequences of the variable fixing may also be taken into account. Sometimes it is advisable to evaluate certain variables in connection with other free variables. In this two step evaluation, the best variables – according to a first evaluation – are temporarily chosen and all free variables are re-evaluated in this context. Constraint evaluations can also be used in these calculations. Detailed examples of different variable evaluations are discussed in Chapter 11.

CHAPTER 10

A NEW SOLUTION FOR THE GENERAL SET COVERING PROBLEM

10.1 Introduction

The theory and applications of the general set covering and set partitioning problem are discussed in the present chapter. Results on these were published in Kovács (1973). The algorithm presented here is closely related to the material of Chapter 9 since it is based on a heuristic method (it can, however, be understood without reading Chapter 9). The background of heuristics is clarified by three lemmata which ensure optimality under some conditions. The procedure is developed further here; it becomes an exact algorithm of branch-and-bound type. But no linear programming is used to calculate the bounds: instead, the heuristic procedure provides the bounds. A computer program is written with a promising result. There are three main reasons for including the set covering problem in the present book. First, it illustrates the good relations between heuristic procedures and exact algorithms, particularly in the case of specially structured problems. Secondly, there are many applications showing the importance of the problem itself. Thirdly, in the general linear integer problems, it may well play an important role – as was briefly discussed in Section 9.3 point (g) and as will be detailed in Chapter 11.

10.2 Problem and terminology

Let us consider the following problem

$$\begin{aligned} &\min \mathbf{c}^T\mathbf{x} \\ &\mathbf{A}\mathbf{x} \geqq \mathbf{e} \qquad (10.1) \\ &x_j \in \{0, 1\} \qquad (j = 1, \ldots, n), \end{aligned}$$

where $\mathbf{A}$ is a given matrix of 0 and 1 elements, $\mathbf{c}$ is an n-vector of positive elements, and $\mathbf{e}$ is an $\mathbf{m}$-vector, each component of which is 1. Problem (10.1) is known as the set covering problem for the following reason. We are given n subsets of the set

$$I_0 = \{1, 2, \ldots, m\},$$

and they are denoted by $I_1, I_2, \ldots, I_n$. A cost c_j is associated with each set I_j. The entire set I_0 has to be covered (each element is chosen at least once) at a minimal cost. In other words a set $J \subset J_0 = \{1, 2, \ldots, n\}$ is to be determined

$$\min_{J \subset J_0} \left\{ \sum_{j \in J} c_j \;\middle|\; \bigcup_{j \in J} I_j = I_0 \right\}.$$

Problem (10.1) is obtained if matrix **A** is defined as

$$a_{ij} = \begin{cases} 1, & \text{if } i \in I_j \\ 0 & \text{otherwise.} \end{cases}$$

The set J is the subscript of variables x_j having the value 1 in the optimal solution of (10.1).

The general set covering problem

$$\begin{aligned} &\min \mathbf{c}^T\mathbf{x} \\ &\mathbf{Ax} \geqq \mathbf{b} \\ &x_j \in \{0, 1\} \quad (j = 1, \ldots, n) \end{aligned} \tag{10.2}$$

can be interpreted in a similar way. The only difference is that element $i \in I_0$ should be covered at least $b_i > 0$ times. (It should be included at least in b_i chosen sets I_j.) Thus **b** is a given m-vector of positive integers.

The set partitioning problem

$$\begin{aligned} &\min \mathbf{c}^T\mathbf{x} \\ &\mathbf{Ax} = \mathbf{e} \\ &x_j \in \{0, 1\} \quad (j = 1, \ldots, n) \end{aligned} \tag{10.3}$$

is the same as (10.1) except that the inequalities are substituted by equations. The same interpretation may also be used for (10.3), the only difference being that each element $i \in I_0$ should be covered exactly once. Each feasible solution **x** of (10.3) defines a partitioning of set I_0. Let

$$J = \{j \mid x_j = 1\}.$$

Then

$$I_0 = \bigcup_{j \in J} I_j \quad \text{and} \quad I_j \cap I_k = \emptyset,$$

$$\text{if } j \neq k \quad \text{and} \quad j, k \in J.$$

Thus the sets I_j $(j \in J)$ give a partitioning of the set I_0. Therefore (10.3) may be stated so as to determine a partitioning of set I_0 at minimal cost.

10.3 Applications

(a) Planning of bus routes

We are given n possible bus routes with an attached cost c_j. There are m bus stops and **A** is the incidence matrix, that is,

$$a_{ij} = \begin{cases} 1, & \text{if route } j \text{ goes through bus stop } i \\ 0 & \text{otherwise.} \end{cases}$$

A bus network of minimal cost is to be determined in such a way that at least one bus route should pass by each bus stop. If variable x_j has the value 1 or 0 depending on whether bus route j is realized or not, then (10.1) is the mathematical model for the bus route planning.

(b) Airline-crew scheduling

Prerisely one crew should be assigned to each of the given m flights in such a way that each crew should obtain an acceptable full assignment (limited number of flights, not too long working time, etc.). The objective is to minimize the number of crews actually needed. To solve this problem let us determine a great number of acceptable crew assignments and calculate matrix **A**

$$a_{ij} = \begin{cases} 1 & \text{if flight } i \text{ is included in assignment } j \\ 0 & \text{otherwise} \end{cases}$$

and the meaning of the variables is

$$x_j = \begin{cases} 1 & \text{if assignment } j \text{ is accepted} \\ 0 & \text{otherwise.} \end{cases}$$

Then a set partitioning problem (10.3) is obtained with $c_j = 1$ $(j = 1, \ldots, n)$.

(c) Switching circuit design

A function $F(u_1, u_2, \ldots, u_N)$ is called a truth function if both the variables $u_1, \ldots, u_N$ and the function F can take only the values 0 and 1. The value 0 is also referred to as the off-position of the switch or the truth value *false*. Similarly the value 1 is also interpreted as the *on* position or truth value *true*. The costs of the AND gate and the OR gate are given. The problem is to realize the truth function $F(u_1, \ldots, u_N)$ at a minimal cost. Let us suppose that the function is given either in a tableau form or in a disjunctive normal form. The latter is the disjunction of different terms, where each term is a conjunction of a subset of variables $u_1, \ldots, u_N$ and their negated form $1 - u_1, \ldots, 1 - u_N$.

Let $\tilde{u}_k$ denote either variable u_k or its negated form $1 - u_k$.

Definition: *The conjunction* $Q = \tilde{u}_{k_1}\tilde{u}_{k_2} \ldots \tilde{u}_{k_r}$ *is a prime implicant of function* F *if* $\tilde{u}_{k_1}\tilde{u}_{k_2} \ldots \tilde{u}_{k_r} = 1$ *implies that* $F(u_1, \ldots, u_N) = 1$ *and no part of* Q *has the same property.*

Then our problem may be transformed into a set covering problem by the following steps:

(i) Determine all prime implicants $Q_1, Q_2, \ldots, Q_n$ of function F and their attached costs $c_1, c_2, \ldots, c_n$. (The number of conjunctions in the prime implicant times the price of the AND gate plus the price of the OR gate counts once because these prime implicants will be connected by the sign of disjunction.)

(ii) Let $\mathbf{u}^1, \mathbf{u}^2, \ldots, \mathbf{u}^m$ denote all the vectors for which $F(\mathbf{u}^k) = 1$, and define the matrix $\mathbf{A}$

$$a_{ij} = \begin{cases} 1 & \text{if } Q_j(\mathbf{u}^i) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

Then the truth function F may be written as

$$F = \sum_{j=1}^{n} Q_j x_j$$

and the problem becomes the set covering problem (10.1).

If the function F takes the value 1 for more than half of the values of the argument, then the function $1 - F$ may be determined instead. There are also other tricks to decrease the size of the problem. Similar methods may be applied to more complicated logical functions than the AND, OR gates. For example, integrated circuits of given function and price can be used.

10.4 Survey of methods

In practice, a version of any integer programming method may be tried out for the set covering problem. On the other hand, since a great many papers have already been devoted to the subject, only a partial list of publications is mentioned here in which each direction is represented by one or two papers.

First of all a version, by Martin (1963), of the Gomory (1963) cutting plane method is reported as being effective for set covering problems. A paper by House et al. (1966) is devoted to the development of a special algorithm solely for the set covering problem. The substance of the method is the construction of additional rows to matrix $\mathbf{A}$ for the exclusion of solutions not better than the best one obtained so far during the algorithm. The paper of Bellmore and Ratliff (1971) also falls within the category of cutting plane methods with a substantially different type of cutting method.

A typical example for the use of the branch-and-bound method is contained in the paper of Lemke et al. (1971). The main difference between their approach and ours (discussed in Section 10.5–10.6) is that they are solving linear programming subproblems to obtain bounds and in the method of the present chapter, no linear programming is used at all. Most probably the structure of branching is also different.

Heuristic methods play an important role in large problems for several reasons. The airline-crew scheduling is solved by a heuristic method by Arabeyre, et al. (1969); Garfinkel and Nemhouser (1969) apply other heuristics for the set partitioning problem. A rounding process and consecutive fixing is used to decrease the size of the problem. The smaller problems so obtained are solved by existing integer programming methods. The group theoretic approach of integer program-

ming is used by Thiriez (1971) having the special advantage that in most cases the determinant of the optimal basis B is usually small because of the 0–1 elements in matrix **A**.

10.5 A heuristic method

The description of the method for solving the set covering problem consists of two parts. The first part gives an explanation of a heuristic method. The second one (next section) describes an exact method using the branch-and-bound principle and also the heuristic method for calculating bounds.

Let us define the set of uncovered rows I_k, and the set of subscripts of unused variables J_k, after executing iteration k. At the beginning, all rows are uncovered and no columns are used

$$I_0 = \{1, 2, \ldots, m\}, \qquad J_0 = \{1, 2, \ldots, n\}.$$

Let us introduce the column count at iteration k as

$$n_j^k = \sum_{i \in I_k} a_{ij}$$

and the evaluation of unused variable at iteration k

$$r_j^k = \begin{cases} \dfrac{c_j}{n_j^k} & \text{if} \quad n_j^k > 0 \\ & \qquad \text{for} \quad j \in J_k \\ \infty & \text{if} \quad n_j^k = 0 \end{cases}$$

r_j^k is nothing other than the row covering cost for each individual row, thus it gives an evaluation of variable j at iteration k.

At iteration $k+1$ the variable $x_{j_{k+1}}$ with the best evaluation is chosen

$$r_{j_{k+1}}^k = \min_{j \in J_k} r_j^k.$$

The new sets are easily calculated

$$J_{k+1} = J_k - \{j_{k+1}\} \qquad \text{and} \qquad I_{k+1} = I_k - M_{k+1},$$

where M_{k+1} is the set of subscripts of the rows covered by variable $x_{j_{k+1}}$

$$M_{k+1} = \{i \mid i \in I_0, \quad a_{ij_{k+1}} = 1\}.$$

The procedure is continued until either I_t or J_t becomes empty. In the first case we have obtained a feasible solution of the problem, in the second no feasible solution exists. Let us suppose that a solution is obtained, that is, $I_t = \emptyset$

$$\hat{x}_j = \begin{cases} 1 & \text{if} \quad j = j_1, j_2, \ldots, j_t \\ 0 & \text{otherwise.} \end{cases}$$

Then the tree lemmata below show how good this solution may be.

$$\hat{J} = \{j_1, j_2, \ldots, j_t\}$$

is the set of indices obtained above and for any $j \in \hat{J}$ form the set

$$H_j = \{i \mid a_{ij} = 1, \quad a_{ik} = 0, \quad \forall\, k \in \hat{J} - \{j\}\}.$$

Furthermore for each $j \in \hat{J}$ let

$$n_j^* = |H_j|,$$

the number of elements in set H_j and

$$r_j^* = \begin{cases} c_j/n_j^* & \text{if} \quad n_j^* > 0 \\ \infty & \text{if} \quad n_j^* = 0. \end{cases}$$

Now we can state

Lemma 10.1: *If*

$$r_j^* \leqq r_k^0 \quad \textit{for any } j \in \hat{J} \textit{ and for any } k \in \{1, 2, \ldots, n\} - \hat{J}, \tag{10.4}$$†

then $\hat{\mathbf{x}}$ *is an optimal solution of problem* (10.1).

Proof: Consider any solution $\tilde{\mathbf{x}}$ of (10.1) and introduce the following additional notation

$$\tilde{J} = \{j \mid \tilde{x}_j = 1\}, \qquad J = \hat{J} \cap \tilde{J},$$

$$r = \min \{r_j^0 \mid j \in \{1, 2, \ldots, n\} - \hat{J}\}. \tag{10.5}$$

Then

$$c^T\hat{x} = \sum_{j \in \hat{J}-J} c_j + \sum_{j \in J} c_j \tag{10.6}$$

$$\sum_{j \in \hat{J}-J} c_j = \sum_{j \in \hat{J}-J} r_j^* n_j^* \leqq r \sum_{j \in \hat{J}-J} n_j^* \leqq r \sum_{j \in \tilde{J}-J} n_j^0 \leqq \sum_{j \in \tilde{J}-J} r_j^0 n_j^0 = \sum_{j \in \tilde{J}-J} c_j. \tag{10.7}$$

If the sum

$$\sum_{j \in J} c_j$$

is added to both ends of the inequality series (10.7), then the desired inequality

$$\mathbf{c}^T\hat{\mathbf{x}} = \sum_{j \in \hat{J}} c_j \leqq \sum_{j \in \tilde{J}} c_j = \mathbf{c}^T\tilde{\mathbf{x}} \tag{10.8}$$

is obtained for any $\tilde{\mathbf{x}}$ feasible solution of (10.1), which proves the lemma.

It should be noted that if there is a number $n_k^* = 0$ ($k \in \hat{J}$), then the corresponding variable x_k may be set equal to zero without changing the feasibility of the

† If we define the numbers r_j^* for $j \notin \hat{J}$ similarly, the statement of the lemma can be sharpened supposing only $r_j^* \leqq r_k^*$ for any $j \in \hat{J}$ and $k = \{1, 2, \ldots n\} \setminus \hat{J}$.

solution. The subscript k should be deleted from $\hat{J}$ and the numbers $n_j^*(j \in \hat{J})$ should be recalculated. Now the assumption of Lemma 10.1 may be checked again. If there are several variables with $n_k^* = 0\ (k \in \hat{J})$ then the order in which they are set equal to zero in the considered solution is important. Different ordering may result in feasible solutions of different objective function value.

The following lemma gives a stronger result:

Lemma 10.2: *Let us define the numbers u_i for each row*

$$u_i = r_{j_s}^{s-1}, \quad \text{if} \quad i \in M_s, \quad \text{but} \quad i \notin M_1, \ldots, M_{s-1} \tag{10.9}$$

$$\text{If} \quad \sum_{i=1}^{m} a_{ij} u_i \leqq c_j \quad \text{for any} \quad j \in \{1, 2, \ldots, n\} \tag{10.10}\dagger$$

then $\hat{\mathbf{x}}$ is an optimal solution of problem (10.1).

Proof: Consider any feasible solution $\tilde{\mathbf{x}}$ of (10.1). Because of the definition of numbers u_i

$$\mathbf{c}^T \hat{\mathbf{x}} = \sum_{i=1}^{m} u_i. \tag{10.11}$$

On the other hand, using definition (10.5), supposition (10.10), and the fact that $\tilde{\mathbf{x}}$ is a feasible solution of (10.1), that is,

$$\sum_{j=1}^{n} a_{ij} \tilde{x}_j \geqq 1 \qquad (i = 1, \ldots, m)$$

we obtain

$$\sum_{i=1}^{m} u_i \leqq \sum_{i=1}^{m} u_i \left(\sum_{j=1}^{n} a_{ij} \tilde{x}_j \right) = \sum_{j=1}^{n} \sum_{i=1}^{m} a_{ij} \tilde{x}_j u_i = \sum_{j \in \tilde{J}} \sum_{i=1}^{m} a_{ij} u_i \leqq \sum_{j \in \tilde{J}} c_j = c^T \tilde{\mathbf{x}} \tag{10.12}$$

Then (10.11) and (10.12) together give the desired result, viz.

$$\mathbf{c}^T \hat{\mathbf{x}} \leqq \mathbf{c}^T \tilde{\mathbf{x}},$$

which proves the lemma.

The following lemma provides a lower bound for the branch-and-bound procedure if Lemmata 10.1 and 10.2 did not prove the optimality of the feasible solution $\tilde{\mathbf{x}}$ obtained by the above heuristic algorithm. The optimality of $\tilde{\mathbf{x}}$ may also be proven sometimes by this result (see the notes after the lemma).

† The inequalites must be satisfied also for $j \in \hat{J}$, i.e. all rows should be covered exactly once in the heuristic solution. In the opposite case all rows covered more than once should be deleted and the covering numbers r and the weights u recalculated. The statement of the Lemma remains true.

Lemma 10.3: *Denote the optimum of problem* (10.1) *by* z^*. *Then*

$$z^* \geqq \sum_{i=1}^{m} f_i = F\,, \tag{10.13}$$

where f_i *is the "minimal covering fraction" of row* i

$$f_i = \min \{r_j^0 \mid 0 < j \leqq n, \quad a_{ij} = 1\} \tag{10.14}$$
$$(i = 1, \ldots, m)\,.$$

Proof: Define the following new problem

$$\min \sum_{j=1}^{n} \sum_{k\ 1}^{m} r_j^0 y_{jk}$$
$$\sum_{j=1}^{n} \sum_{k=1}^{m} g_{ijk} y_{jk} \geqq 1 \qquad (i = 1, \ldots, m) \tag{10.15}$$
$$y_{jk} \in \{0, 1\}$$

where
$$g_{ijk} = \begin{cases} a_{ij} & \text{if} \quad i = \mathrm{k} \\ 0 & \text{otherwise.} \end{cases}$$

For any feasible solution $\mathbf{x}$ of (10.1), there exists a solution of (10.15) in such a way that the objective function values are the same, e.g.

$$y_{jk} = a_{jk} x_j \qquad (j = 1, \ldots, n; \quad k = 1, \ldots, m)\,.$$

On the other hand, the minimum of (10.15) is obviously F. This proves statement (10.13) of the lemma.

Notes: 1° If $\mathbf{c}^T\mathbf{x} = F$, then $\hat{\mathbf{x}}$ is an optimal solution of (10.1).

2° If $\mathbf{c}^T\mathbf{x} > F$, then F gives a lower bound of the optimum of (10.1) which may be used in a branch-and-bound procedure.

10.6 An exact solution for the general set covering problem

A branch-and-bound procedure was developed for solving (10.1). As this type of method has been extensively discussed in Chapter 3, only a few characteristics and special properties of the present computer program are mentioned here

1° The variable for branching is chosen in the following way. Let us determine subscript i_0 by the minimization

$$\min_{i} \sum_{j=1}^{n} a_{ij} = \sum_{j=1}^{n} a_{i_0 j}$$

and let

$$K = \{j \mid a_{i_0 j} = 1, \quad 1 \leqq j \leqq n\}.$$

A simple lower bound calculation is then made for all hypothetical fixings $x_j = 0$ and 1 ($j \in K$). The lower bounds are denoted by L_j^0 and L_j^1. Then variable x_{j_0} is chosen

$$\max |L_j^0 - L_j^1| = |L_{j_0}^0 - L_{j_0}^1|.$$

After each variable fixing the better of the newly formed branches is chosen if at least one of them is feasible. In the opposite case the best branch is taken the fast reconstruction of which is provided.

2° Fast termination of subproblems is guaranteed if no solution exists.

3° In the case of abnormal termination usually good feasible solutions are available.

4° Lower bounds are provided by the heuristic procedure described in Section 10.5, together with the corresponding solutions which often turn out to be optimal (see Lemmata 10.1–10.3).

The computer code of this algorithm was written in FORTRAN IV, for a CDC 3300 computer. Several problems with up to 100 variables were solved, each of them in a few seconds. The program is being tested and further developed for larger problems, in which case it will be necessary to retain another representation of the coefficients in the core. A slightly modified version of the program is capable of solving (10.2) without transforming it into the form of (10.1). A similar approach might be used to solve (10.3), but most probably a different evaluation of the variables and other substantial modifications would give a more effective algorithm. It should also be mentioned that many effective simplex type methods have been worked out for (10.3) because it has equation-type constraints.

CHAPTER 11

COMPLEX ALGORITHMS

11.1 Introduction

The main purpose of this chapter is the design of complex algorithms for computers. The first few sections contain the main tools of such algorithms. Some of them are only mentioned here with reference to chapter number; others are developed here. Some of the possible combinations of different algorithms are then outlined. One section aims at the man-machine interaction, this implies the use of prior information, series of interrupted and revised runs in the case of very large problems, the change of control parameters by manual intervention, etc. One particular section is devoted to the actual design of discrete programming procedures; in this section the main functions of the central organization of complex algorithms are described. Finally, an outline is given of the experience that has been gained utilizing a complex algorithm of the type described in the foregoing sections. Most of the tools and ideas of this chapter were tested on this algorithm, except a few obvious, general notes on the mixing of different algorithms which have occurred throughout the preparation of the present work. A short review of the numerical experience gained is also included in the last section.

The framework of the procedure is mainly the one described in Chapter 2, but substantial modifications are introduced in order to obtain greater flexibility and efficiency.

11.2 Heuristic procedures for obtaining feasible solutions

Many of these procedures were discussed in Chapter 9. They are used mainly to determine initial feasible solutions or sometimes other feasible solutions during the steps of the main procedure. They have three main functions. First, the introduction of the objective function constraint (permitting only the solutions better than the best one obtained so far) usually accelerates the main algorithm. Secondly, the heuristic procedures usually give estimations for the exact optimum both for the original problem and for the subproblems obtained by fixing some of the variables. These bounds may be used to control the central organization (especially in the procedures of branch-and-bound type, but also of others) and to avoid costly runs which improve the best solution obtained by only a negligible amount. Thirdly, the heuristic procedures usually provide some good feasible solutions in the case of very large problems (see also Section 11.10).

11.3 *s*-Constraints

s-Constraints were defined and extensively discussed in Chapter 5. They are nonnegative linear combinations of the original inequality constraints. If the multipliers are carefully chosen, then very strong constraints may be obtained, one or a few of which may substitute the original constraints to a great extent. They may provide lower bounds, obligatory variable fixings, infeasibility of certain pseudo-solutions, and sometimes hints for good variable evaluations. Several methods for calculating good *s*-constraints were discussed in Chapter 5. Another one is used in the algorithm described in Section 11.13. This is an adaptive approach in which the multipliers are obtained by counting the number of infeasibilities for each constraint (see Chapter 9). The past infeasibilities must have a decreasing importance, therefore these weights are normed (divided by a relatively large number) from time to time. This division also helps to avoid overflow in the computer. The frequency of division is important because it determines the "forgetting procedure".

11.4 Tests

Such tests may be defined as sufficient conditions for obligatory variable fixings and as a means of showing the infeasibility of certain pseudo-solutions. A detailed discussion is to be found in Chapter 5. It is very important to realize that neglecting the tests and carrying out long unnecessary searches is just as great a fault as their constant application all the time for all the constraints. Besides the tests generally applicable, it is very useful to determine special tests for special classes of structured problems. Both questions will be detailed in Section 11.11.

11.5 Free variable choice rules

When no obligatory variable fixing can be determined, then it is very important which free variable is chosen next. In the case of better choices we may obtain a feasible solution much earlier, and the enumeration algorithm is accelerated by the stronger objective function constraint. Another advantage is that the opposite fixing of the really good variables (at zero value) is usually enumerated very quickly. Thus the earlier position it stands the better because it corresponds to the enumeration of a larger solution set. In brief, the importance of the order of variables can be demonstrated by the fact that if a variable precedes another by k positions, then the ratio of the number of elements in the corresponding pseudo-solutions is 2^k.

Let us consider now the following problem

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

$$\sum_{j=1}^{n} a_{ij}x_j \geqq b_i \qquad (i = 1, \ldots, m) \tag{11.1}$$

$$x_j \in \{0, 1\} \qquad (j = 1, \ldots, n),$$

where $c_j \geqq 0$ $(j = 1, \ldots, n)$ can be supposed without restricting the generality. Furthermore, let us denote the subscript of variables already fixed by

$$J = \{j_1, j_2, \ldots, j_k\}$$

the subscript set of free variables is denoted by F. Then

$$J \cap F = \emptyset \text{ and } J \cup F = \{1, 2, \ldots, n\}.$$

The present values of fixed variables are denoted by $\delta_{j_1}, \ldots, \delta_{j_k}$. The (11.1) is transformed as usual

$$\min \sum_{j \in F} c_j x_j + K$$

$$\sum_{j \in F} a_{ij}x_j \geqq \bar{b}_j \qquad (i = 1, \ldots, m) \tag{11.2}$$

$$x_j \in \{0, 1\} \qquad (j = 1, \ldots, n),$$

where

$$\bar{b}_i = b_i - \sum_{j \in J} a_{ij}\delta_j, \quad K = \sum_{j \in J} c_j \delta_j. \tag{11.3}$$

We are looking for a free variable to fix at value 1 in the present ordered pseudo-solution

$$\phi = (x_{j_1} = \delta_{j_1}, \ldots, x_{j_k} = \delta_{j_k}).$$

For this purpose an evaluation of free variables should be determined for the transformed problem (11.2). We can suppose that at least one of the inequalities $\bar{b}_i \leqq 0$ $(i = 1, \ldots, m)$ does not hold, because in the opposite case $x_j = 0$ $(j \in F)$ is obviously an optimal solution of (11.2).

Balas (1965) suggested the calculation of the following value

$$v_j = \sum_{i=1}^{m} (\bar{b}_i - a_{ij})^+ \qquad (j \in F), \tag{11.4}$$

where the superscript $+$ denotes the positive part. Namely, if the variable fixing $x_j = 1$ is executed, then $(\bar{b}_i - a_{ij})^+$ is the infeasibility in constraint i. Thus the quantity v_j is a natural measure of infeasibility after the variable fixing $x_j = 1$. $v_j = 0$ denotes a feasible solution. In any case $x_{j_{k+1}} = 1$ is chosen, where

$$\min_{j \in F} v_j = v_{j_{k+1}}.$$

If $v_j > 0$ $(j \in F)$, that is, no feasible solution can be obtained by fixing just a single variable, then there are several other possibilities for evaluating the free variables.

Our algorithm uses a modified version of the above Balas value in order to indicate also the price of the improvement in the infeasibility

$$w_j = \left\{ \sum_{i=1}^{m} (\bar{b}_i - a_{ij})^+ - \sum_{i=1}^{m} \bar{b}_i^+ \right\} \Big/ (\varepsilon + c_j) \qquad (j \in F), \tag{11.5}$$

where $\varepsilon > 0$ is a small number relative to the coefficients c_j to avoid the zero divisor. In other words, w_j expresses the feasibility improvement obtained for unit cost if $x_j = 1$ is chosen next.

Further variants may be obtained by considering the difficulty of each constraint. This is done by introducing different nonnegative weights s_i for the constraints

$$u_j = \left\{ \sum_{i=1}^{m} s_i[(\bar{b}_i - a_{ij})^+ - \bar{b}_i^+] \right\} \Big/ (\varepsilon + c_j) \qquad (j \in F). \tag{11.6}$$

There are several possibilities for determining these weights. One way is the use of coefficients in row i, for example,

$$s_i = \bar{b}_i^+ / \sum_{j \in F} a_{ij}^+ \qquad (i = 1, \ldots, m) \tag{11.7}$$

or the minimal cost may be assigned to row i by which it can be satisfied, etc.

We have chosen a different, adaptive approach. The frequency of infeasibility is counted for each row, similarly to the way described in Section 11.3. The method does not give the measure of infeasibility of the s-constraint because the weights are changing continually and the s-constraint is recalculated only after several iterations.

Another, more complicated variable evaluation is also used, but only at the beginning of the algorithm when the number of free variables is large. The reason is that a relatively large amount of calculation is rewarding only if the present pseudo-solution is large enough.

The basic idea is that each free variable is temporarily fixed at value 1 and continued by k further variable fixings according to any variable evaluation. The free variable is chosen for permanent fixing for which the above continuation gives the best result, according to feasibility and cost. One possible realization of this idea may be described as follows.

Denote by u_j $(j \in F - \{r\})$ an evaluation of free variables after the variable fixing $x_r = 1$. Let us order these values

$$u_{p(1)} \geqq u_{p(2)} \geqq \ldots .$$

The second level evaluation of variable x_r may be determined by the best infeasibility value obtained in at most k further variable fixing

$$h_r = \min_{t \leqq k} \left\{ \sum_{i=1}^{m} \left(\bar{b}_i - a_{ir} - \sum_{s=1}^{t} a_{ip(s)} \right)^+ \;\middle|\; K + c_r + \sum_{s=1}^{t} c_{ip(s)} \leqq z^* - 1 \right\}, \tag{11.8}$$

where z^* is the objective function value corresponding to the best feasible solution obtained so far (if one has already been found; in the opposite case $z^* = +\infty$). These values h_r are calculated for each $r \in F$ and the smallest one is chosen for permanent fixing. This method may also be combined with more complicated primary evaluations like the ones described by (11.5) and (11.6) instead of using the simplest one, viz. (11.4).

It is important to note that actually not all of the free variables are evaluated, but they are restricted to a relatively small subset. The method of restricted enumeration is described in the next section.

11.6 Restricted enumeration

Let us consider the ordered pseudo-solution

$$\phi = (x_{j_1} = \delta_{j_1}, \ldots, x_{j_k} = \delta_{j_k})$$

obtained when the algorithm was applied to (11.1). Some of the fixed variables may also be tied. J and F denote the subscripts of fixed and free variables respectively, and $\bar{b}_i$ stands for the transformed right hand side as in Section 11.5. Let us suppose that a free choice of variable fixing is to be made. (No tests are applicable.) It would be very useful to restrict the choice to the smallest possible subset of F. This would have several advantages. First of all, the enumeration is restricted to a fraction of the total number of possibilities, proportionally to the decrease in the number of candidates. Secondly, fewer variables need be evaluated. And last but not least, the choosing of "essential" variables as early as possible usually accelerates the algorithm in both branches. When it is fixed at 1, feasible solutions are found earlier. And when it is fixed at zero, the pseudo-solution often turns out to be infeasible.

Though the basic idea has already been mentioned in connection with the set covering problem in Chapter 10, the approach used in the present algorithm is substantially different. Let us consider any of the constraints not satisfied by ϕ^0 (the free variables are considered as being zero), that is,

$$\sum_{j \in F} a_{ij} x_j \geqq \bar{b}_i > 0. \tag{11.9}$$

Obviously at least one of the variables, having a subscript from the following set, must be chosen

$$\{j \mid a_{ij} \geqq 0, \quad j \in F\}. \tag{11.10}$$

We can find an even smaller subset with the same property. For this purpose let us order the coefficients standing on the left hand side of inequality (11.9)

$$a_{ip(1)} \geqq a_{ip(2)} \geq \ldots \geqq a_{ip(s)} > 0, \tag{11.11}$$

and determine the number r in such a way that

$$\sum_{q=r}^{s} a_{ip(q)} \geqq \bar{b}_i > \sum_{q=r+1}^{s} a_{ip(q)}. \tag{11.12}$$

It is easy to see that the subscript set

$$\{p(1), p(2), \ldots, p(r)\} \tag{11.13}$$

can play the role of set (11.10), at least one of the corresponding variables should be fixed at value 1. Let us denote this set by J_i. This set may be calculated for each row with $\bar{b}_i > 0$. If $\bar{b}_i \leqq 0$, then let $J_i = \emptyset$. A simple choice among these sets is

$$|J_{i_0}| = \min \{|J_i| \mid 1 \leqq i \leqq m, |J_i| > 0\}. \tag{11.14}$$

If the number of elements is the same in several of these sets

$$|J_i| = |J_{i_0}| \qquad \text{for any} \qquad i \in I_0,$$

then the choice rule is applied to variables

$$x_j : j \in \bigcup_{i \in I_0} J_i$$

and if the variable fixing $x_{j_0} = 1$ is accepted, then one of the sets J_i containing the subscript j_0 is used for restricted enumeration. The frequency of occurrence of the different subscripts in sets J_i may also be an indication of importance; this may also be used in a combined choice rule.

These sets J_i may also be used for an infeasibility test. Let us consider a pairwise disjoint subset of $J_1, \ldots, J_m$,

$$J_r \cap J_s = \emptyset \qquad \text{if} \qquad r \neq s \qquad \text{and} \qquad r, s \in I \subset \{1, \ldots, m\}$$

and

$$\bigcup_{r \in I} J_r \neq \emptyset.$$

(The set I may contain only a single element.) Furthermore let z^* denote the objective function value of the best feasible solution found so far. Now if

$$\sum_{j \in J} c_j \delta_j + \sum_{r \in I} \min_{j \in J_r} c_j \geqq z^*, \tag{11.15}$$

then $\{\phi\}_s = \emptyset$, our present pseudo-solution contains no feasible solution. (The objective function constraint is also considered!)

Test (11.15) can easily be generalized if other sets of variable subscripts are formed out of which at least k elements are to be chosen. Let us consider again a constraint (11.9) and the ordered coefficients (11.11). If the numbers r and k are determined in such a way that $k \leqq r$ and

$$\sum_{q=r}^{s} a_{ip(q)} \geqq \bar{b}_i - \sum_{q=1}^{k-1} a_{ip(q)} > \sum_{q=r+1}^{s} a_{ip(q)},$$

then at least k of the variables corresponding to the subscripts

$$\{p(1), \ldots, p(r)\}$$

should be fixed at value 1. A more general test of type (11.15) can easily be formulated. It should be noted that restrictions of different strength may be obtained for one particular row if k takes different values. Naturally the value of k is usually different for each row. Similar restriction sets may be obtained for nonnegative linear combinations of rows presently not satisfied and a parametric analysis may be carried out for the multipliers. A substantial reduction in computer runs has been obtained for all larger problems when this procedure has been applied. (The running time was smaller by a factor of 4–6.)

11.7 More flexible enumeration procedures

(a) Restarting

The idea of enumeration algorithms was presented in Section 2.3 where the method of Lawler and Bell was described. A substantially more flexible general framework of enumeration procedures was discussed in Section 2.5. Practically any order of variable may be considered and any of the free variables may be chosen for fixing. This means a full flexibility at the forward steps. But when a backward step is taken, there is no possibility of reordering the variables according to the procedure of Section 2.5. In other words the variable fixings, made earlier, will be eliminated later. No matter how carefully the fixings were made, in the new situation – fixing several further variables – the picture may be entirely different and some variables which seemed to be good ones earlier may become less favourable. Furthermore, as the procedure goes on we obtain further information on the problem. For example, new feasible solutions may be found (thus the objective function constraint gets stronger), the learning procedures built in may have very good evaluating numbers both for the constraints and for the variables, etc. As a consequence, the previous variable fixings may become a burden. One may well find oneself saying "I wish I'd started the whole thing again, I'd do it much better". Naturally such restarting might really be advantageous if all the information obtained so far were to be utilized – as practice shows. But if we restart the entire algorithm from the very beginning we are also losing something, viz. we have to enumerate again all that has been done so far; in which case it should not be done very often. It is also possible that sometimes only partial restarting is applied. But in any case, it is quite difficult to compare the advantage of greater freedom in choice with the loss of recalculations.

(b) Permutation of untied variables

There are other methods where the increased flexibility does not accompany any recalculation. An idea of this type is described by Loehman et al. (1970), and will

be adapted to our general procedure of Section 2.5. The substance of these modifications may be described shortly as follows.

When the backtracking step is executed in the algorithm of Section 2.5 (Step 6), then not only the last untied variable is considered, but also all the previous ones until the next tied variable, and one of those may be chosen freely to change it to the opposite value. The modified text of this step can be described as

Step 6: Determine the last untied variable and eliminate all fixings following this term. One of the variables, not followed by any untied variable, is selected and put to the last place. This variable is fixed and tied at the opposite value.

Example: Consider the following ordered pseudo-solution

$$\phi = (5, 7, \underline{\bar{1}}, 9, 3, 11, \underline{\bar{4}}, \underline{12}, \underline{16}).$$

The last tied variables are deleted first

$$\phi = (5, 7, \underline{\bar{1}}, 9, 3, 11).$$

Now any of the variables 9, 3, 11 may be chosen, e.g. x_9, then

$$\phi = (5, 7, \underline{\bar{1}}, 3, 11, \underline{\bar{9}})$$

is the result of the backtracking.

It should be noted that when a variable is chosen from among the above described untied ones, a variable evaluation similar to those described in Section 11.5 may be used. Now, obviously the effect of changing variable x_j from the present value (usually 1) to the opposite one should be considered.

The justification for this change in the algorithm is quite easy. For this purpose let us consider a general ordered pseudo-solution

$$\phi = (\alpha_1, \ldots, \underline{\alpha_p}, \alpha_{p+1}, \ldots \alpha_q, \underline{\alpha_{q+1}}, \ldots, \underline{\alpha_r}),$$

where α_k represents a variable fixing of type $x_j = 1$ or $x_j = 0$ and the underlinings stand for the tying of corresponding variable fixings. If a permutation of elements $\alpha_{p+1}, \ldots, \alpha_q$ is taken, then this corresponds to another realization of our algorithm – but there is no difference between the two of them as far as the consequences are concerned (none of the expressions $\alpha_{p+1}, \ldots, \alpha_q$ is tied!), and only these consequences (variable tyings) can exclude any solution. In other words if a permutation of elements $\alpha_{p+1}, \ldots, \alpha_q$ substitutes the original order, then the same solution sets are considered, i.e. the original proof of the algorithm is still valid. In some cases, it may be reasonable to go deeper into the untied variables than is permitted by the last untied variable. For example if

$$\phi = (2, \underline{3}, 11, 9, 7, \underline{8}, 12, 17, \underline{1}, \underline{\bar{6}}, \underline{13})$$

then the last three variables are omitted:

$$\phi = (2, \underline{3}, 11, 9, 7, \underline{8}, 12, 17)$$

and one of the variables 12 and 17 may be chosen at the backtracking. If the opposite fixing of variable 11, 9 or 7 is a very good choise, then it can be done at the price of losing the calculation made for the ordered pseudo-solution

$$\phi = (2, \underline{3}, 11, 9, 7, \overline{8}),$$

because the backtracking with variable 9, for example, must imply the deletion of the tying sign for variable 8

$$\phi = (2, 3, 11, 7, 8, 12, 17, \underline{9}).$$

It is very easy to describe and prove this modification.

(c) Inserting tied variables

When a new feasible solution has been obtained and a stronger objective function constraint is thereby used or a new *s*-constraint is generated, then the present variable fixings may imply further ones. It can easily happen that the above mentioned new constraints would already show these implications for a part of the fixed variable. In other words, it is possible that the newly fixed and tied variable is not placed last, but inserted somewhere in the middle. In this way the enumeration is shortened.

(d) Neighbourhood search

Though the heuristic procedures are widely applicable at almost any part of the enumeration algorithms, one feature seems to be especially useful in connection with the flexible backtracking. Namely, if the leaving out of any variable from the last group of untied variables results in a feasible solution, then it is recognized by the procedure described in point b of the present section. But if only a strongly improved infeasibility measure is obtained, then it may be worth carrying out a heuristic search for feasible solutions by leaving out individually, pairwise, etc. the fixed variables. This search may be extended to the fixed variables not in the last untied group. These heuristics, if they are not exaggerated, require only little additional time and ensure that the enumeration, going through the neighbourhood of a feasible solution, will find that solution.

Example: for inserting a tied variable. If a new constraint is used to test the ordered pseudosolution

$$\phi = (\underline{1}, \overline{\underline{5}}, \underline{6}, 2, 5, 11, 7)$$

and it turns out, that the fixings $\bar{8}$ and 12 are consequences, then the subsets of ϕ are also examined and it may turn out, for example, that the following implications hold true

$$(\underline{1}, \underline{\bar{5}}) \rightarrow 12 \qquad \text{and} \qquad (\underline{1}, \underline{\bar{5}}, \underline{6}, 2) \rightarrow \bar{8},$$

then instead of the ordered pseudo-solution

$$\phi = (\underline{1}, \underline{\bar{5}}, \underline{6}, 2, 5, 11, 7, \underline{\bar{8}}, \underline{12})$$

the following one, representing a more advanced state of enumeration, may be used

$$\phi = (\underline{1}, \underline{\bar{5}}, \underline{12}, \underline{6}, 2, \underline{\bar{8}}, 5, 11, 7)$$

It is easy to show that this inserting method does not effect the correctness of the algorithm of Section 2.5. This technique of insertion may also be used to avoid heavy overtesting (see Section 11.11).

11.8 An adaptive approach to subproblems of small size

It is quite obvious that the same amount of testing after many variable fixings may be a heavy burden. In other words the time spent in testing should not exceed the time needed for complete enumeration. When the number of free variables is only 4–6, then no testing is necessary. The idea may be developed further and a special routine for relatively small sized problems may be developed. But it is very hard to guess the range, i.e. the number of free variables, in which it is efficient. To overcome this difficulty an adaptive approach was used. The time used by this special routine is always measured. Several time intervals were defined. The routine starts working if the number of free variables is smaller than n_s. If the running time is less than t_1, then n_s is increased by 1. In the time interval t_1, t_2 there is no change in n_s and in further intervals n_s is decreased by larger and larger numbers. If the subproblem is not solved within a given time t_k, then the run is interrupted and the central organization gets the control again.

It is interesting to note that the maximal number of variables in the subproblems has been changing slowly between 15 and 35 (see also Section 11.12). This shows that the adaptive approach was very useful because sometimes relatively large subproblems were solved easily; sometimes substantially smaller problems were more difficult. On the other hand, usually quite a significant amount of computer time – otherwise wasted on heavy testing – was saved. Interruption of the subrutine, because of its last time limit, and thus wasted time, happened very rarely.

At present, a simplified version of the original algorithm is used for problems of small size. Further research should be done to develop procedures which are usually very fast for smaller problems even if, on occasion, they would take an extremely long time. The heuristic algorithms of Chapter 9 may also be utilized. In particular, the relaxation and some of the bounding techniques seem to be very attractive.

11.9 Mixing of algorithms

An almost unlimited number of possibilities are available for improving algorithms of discrete programming by mixing them with each other. Although more than two different procedures may also used in one algorithm, we shall briefly discuss only some of the pairwise relations. The wide variety of heuristic algorithms can give great help (see Chapter 9) but for simplicity we shall disregard them in this section.

First of all, if the main algorithm is a single branch enumeration or a branch-and-bound method, then the smaller size subproblems may be solved by any of the existing methods (see Section 11.8).

(a) Single branch enumeration methods

Such methods may be modified by using a kind of bounding technique. The procedure of enumeration may be interrupted any time, where – according to certain indicators – it is highly unlikely to get good feasible solutions. Then a backtracking step is taken and the ordered pseudo-solution, where the "illegal" backtracking was performed, is saved. At the end of the algorithm these gaps are reconsidered, most probably in much less time than they could have been at their places because an optimal or near optimal solution is known and possibly other information has been gained on the problem (learning routines).

(b) Branch-and-bound techniques

Branch-and-bound techniques are usually coupled with other methods resulting in good branches. For example, many of the heuristic methods also provide lower bounds. The most important group is that of the relaxation methods (see Chapter 9).

Furthermore, there are several reasons why single branch enumeration should be mixed to most of the branch-and-bound procedures. First of all, the number of branches usually increases rapidly and exhausts the core of the computer. Before this happens, some of the already existing branches have to be searched by a memory saving oriented procedure, e.g. a single branch enumeration. Secondly, the accuracy of bounds largely depends on how far we have gone down in the branching. In other words, some of the branches close to the origin of the pseudo-solution tree have small lower bounds only because of the inaccuracy of calculations. This means that it is worth choosing smaller pseudo-solutions, even with somewhat higher lower bounds. Thirdly, jumping from one branch to another, usually results in a lot of computation, necessary for reconstructing the problem at the new branch. This should be avoided so far as is reasonable taking into account the lower bound differences. And finally, especially for larger problems, it is advisable to go down in the pseudo-solution tree rapidly to find feasible solutions. This provides acceptable solutions in the case of interrupted runs, and good feasible solutions usually accellerate most discrete programming algorithms.

(c) Cutting planes

Gomory's cutting plane method was the first exact algorithm for solving discrete programming problems. Even though it does not seem to be practical for larger problems, its influence is still considerable. The method was briefly outlined in Chapter 5. Detailed discussion may be found in Gomory (1963) and Dantzig (1963). Since that time, many other cutting plane methods have been invented. A very interesting direction may be found in Balas (1971) Balas et al. (1971), Glover (1970), Young (1971). A quite recent paper on the subject is that of Burdet (1973). These cutting plane methods are independent procedures but their basic cuts may be applied to accelerate other types of algorithms.

11.10 Man–machine interactions

Human knowledge and experience may be utilized in the case of very large problems to get reasonably good solutions or to accelerate the algorithm and obtain the exact optimum.

(a) Prior information

Prior information from practice or from previous runs may suggest a very likely value for certain variables. Then the enumeration may be started with the ordered pseudo-solution containing these likely fixings in the order of their likeness. These variables should not be tied to ensure completeness of the enumeration. At first, this kind of special start may seem to be dangerous because it determines the order of enumeration to a great extent, and if the guess is not correct, feasible solutions are found much later. But if the flexible backtracking of Section 11.7 is used, then inconvenient fixed variables may be eliminated sooner. Prior information may also be obtained from the knowledge of several feasible solutions which may have been generated by the heuristic part of the algorithm.

(b) An interrupted single branch enumeration

An enumeration of this type may be controlled by hand. The basic idea was briefly outlined in Section 11.9a. The only difference is that after each part, the computer run is interrupted and the examiner decides which pseudo-solution should be considered next. These interruptions make other checks possible. For example, the feasible solutions may be thoroughly examined to determine whether they are really usuable, otherwise a revision of the model has to be done. Furthermore, this analysis may include the understanding of the meaning of variables fixed and, thus, new possibilities may be tried. Although heuristic methods may also be applied in the meantime, they should be separated from the exact algorithm in order to know what is certain and what is merely a possibility.

11.11 Design of algorithms for discrete programming

In the foregoing sections, many ideas were discussed for the acceleration of the algorithm. The previous section outlines a few possibilities when the inspection of the run would help to direct it in the future. In the present section, the main procedures and an authomatic supervisor program of a quite general discrete programming algorithm are summarized. Though it is described mainly in the framework of the algorithm in Section 2.5, most ideas are more generally applicable.

Now we shall discuss the main functions of the supervisor program. First of all, for each ordered pseudo-solution a decision has to be made with regard to the step to be applied:

- heuristic search,
- construction of new s-constraint(s),
- testing,
- switching to a special method for solving the present subproblem,
- choice of free variable for fixing,
- backtracking.

If backtracking is chosen without proving the emptiness of the present pseudo-solution, then it should be stored for later checking.

Secondly, when the function of the next step is decided, there are usually several alternative choices. These are not detailed here as they were already discussed in the foregoing sections.

The basis for these decisions are the prior (built in) and obtained experience with the particular problem. All routines should be carefully examined to see how much time it takes and what the result was. The result often means how large a part of the solution-tree has been eliminated. Besides all this, the supervisor program should take into account the present situation: the number of free variables, the likelihood of obtaining a feasible solution, the strength of constraints, etc.

Prior experience plays an important role in the above choices but since each problem has a special structure, it is important to obtain information for the particular problem. This may be done, for example, by certain learning routines, some of which were described in Section 11.5. Measuring the running time of each routine is necessary in all cases.

One further important point should be emphasized. Each of the above functions is necessary, but if any of them is exaggerated – even if in a part of the algorithm – then only the computer time is wasted. In other words their balanced application is necessary. As an example, if heavy testing is applied all the time, then often no results (consequences) are obtained. No matter how complicated the tests we are using, if we have several substantially different feasible solutions in our present pseudo-solutions, then probably no further obligatory variable fixing can be found.

11.12 Computer experience

An enumeration algorithm with most of the features outlined in the present chapter has been programmed for a CDC 3300 computer. It was written in FORTRAN IV, and consists of approximately 3000 statements.

Some of the routines and their performance will now briefly reviewed.

The algorithm is usually started with an extensive heuristic routine which is based on a neighbourhood search and utilizes a variable evaluation of type (11.6). As the procedure is adaptive, it is repeated a few times with those new weights available at present. In most cases, several good feasible solutions have been found in a few seconds, and the corresponding objective function was usually not further from the optimum than ten per cent.

An s-constraint was generated and updated after about 10 new variable fixings. The adaptive method of Section 11.3 was used. Most of the tests described in Chapter 5 were used, but mainly for the s-constraint. Though some of the tests required only a negligible amount of computer time, the administration, the ordering of coefficients, etc. are time consuming. The constant reordering could have been solved at a price of higher central memory requirement. In the program, except in the case of s-constraints, testing is rarely used for the constraints. This idea can be extended with the tied variable inserting method of Section 11.7c.

Several variable evaluation methods of different complexity are used. At the beginning the adaptive procedure of considering probable future fixings is used; this is described at the end of Section 11.5. At other places, the evaluations (11.6), (11.5) and sometimes (11.4) are used depending on the importance of the decision, which is a function of the number of free variables, strength of constraints (especially the objective function constraint), etc.

The restricted enumeration method, briefly outlined in Section 11.6, is extensively used in a more complicated way. Several sets of variables are calculated for each position, one of which has to be chosen. A complex decision rule is used and the corresponding set is saved and shifted in the case of backtracking. For this purpose, a two-dimensional array would have been required. As the maximal number in one set may be quite large, it would have been very restrictive for the number of parallel sets existing for different positions. A one-dimensional version is used instead, with the following structure

$$K(L), K(L+1), \ldots, K(M),$$

where

$$K(L) = K(M) = M - L + 1,$$

i.e. the length of this block. $K(L+1)$ is the position number and $K(L+2), \ldots, K(M-1)$ are the variable subscripts to choose from. The negative sign means that the corresponding variable has already been tried.

Example: Consider the following ordered pseudo-solution obtained in an enumeration

$$\phi = (5, 3, \underline{2}, 11).$$

As an example for the restricted enumeration, see the vector

$$(6, 1, -5, 8, 12, 6, 5, 2, 2, -3, 5, 7, 4, 8, -2, -11, 5, 7).$$

As the first and last element of each sublist show the number of elements (for forward and backward search), it is easy to separate the three parts

$$(6, 1, -5, 8, 12, 6), \quad (5, 2, 2, -3, 5), \quad (7, 4, 8, -2, -11, 5, 7).$$

The last group, for example, means that it refers to position 4, where x_2 and x_{11} have already been tried and further possibilities are variable fixings $x_8 = 1$ or $x_5 = 1$. If these fixings have been tried, then a backtracking step may be executed without any further free variable choice.

The flexible tree search method of Section 11.7 is optional in the program. Its performance is very good. It is essential if prior knowledge or guess determines a starting ordered pseudo-solution, because the order of these variables otherwise determines the enumeration to a great extent, though they were not really evaluated by thorough calculations.

The small-sized subproblems are handled by what is usually a fast method of single branch enumeration. The initial value was generally 25, i.e. subproblems with less than 25 variables were treated by this special routine. This size has been modified by the adaptive control routine described in Section 11.8. The maximal size of "small problems" remained usually between 20 and 28. Rarely did it go down to 15 or up to 35. The "normal" performance time, when no change of this size was performed, was set between 0.5 and 1.0 second.

The input of the computer program consists of the data of the problem (coefficient matrix **A**, cost vector **c**, right hand side vector **b**), which are converted to standard form first. Following this, several parameters influencing the performance and printing of each routine may be given; The input is completed with the variables likely or unlikely in the optimal solutions, which may speed up the algorithm. The output is controlled by certain input parameters. Feasible solutions of strictly decreasing objective function value are printed out unconditionally. A small additional printing is advisable in order to follow the enumeration because if the run is interrupted (e.g. the allowed time has expired), the present ordered pseudo-solution is sufficient to reconstruct the necessary quantities and continue the calculations where they were stopped. The restricted enumeration is either reconstructed if the corresponding array is also saved and fed back, or new sets are constructed. Practically no computation is lost when the run is restarted.

Several problems of up to 50 rows and 100 variables have been solved, each of them in less than 10 minutes. Good feasible solutions were known after 5–10 seconds. Without substantial change, the declaration may be extended up to

$$mn \leqq 10\,000$$

remaining in the core, where m is the number of constraints and n is the number of variables. The algorithm is being further tested and developed.

REFERENCES TO CHAPTERS 1–11

Abadie, J., ed. (1967) *Nonlinear Programming*, North Holland, Amsterdam.

Abadie, J., ed. (1970) *Integer and Nonlinear Programming*, American Elsevier, New York.

Abadie, J. (1971) "Une Méthode de Résolution des Programmes Non-Linéaires Partiellement Discrets sans Hypothèse de Convexité", *Rèv. Fran. d'Informatique et de Recherche Opèrationelle, 5*, V-1, 23–38.

Agin, N. (1966) "Optimum Seeking with Branch-and-Bound", *Management Science 13*, B176-B185.

Alcaly, R. E. and Klevorick, A. K. (1966) "A Note on the Dual Prices of Integer Programs", *Econometrica, 34*, 206–214.

Armour, G. C. and Buffa, E. S. (1965) "A Heuristic Algorithm and Simulation Approach to Relative Location of Facilities", *Management Science, 9*, 294–309.

Aronofsky, J., ed. (1969) *Progress in Operations Research, Vol. 3:* Relationship between Operations Research and the Computer, John Wiley and Sons, New York.

Ashour, S. and Char, A. R. (1971) "Computational Experience on 0–1 Programming Approaches to Various Combinational Problems", *J. Op. Res. Soc. Japan, 13*, 78–107.

Avi-Itzhak, B., ed (1971) *Developments in Operations Research*, Gordon and Breach, New York.

Balas, E. (1965) "An Additive Algorithm for Solving Linear Programs with Zero-One Variables" *Operations Research, 13*, 517–546.

Balas, E. (1964-1) "Un algorithme additif pour la résolution des programmes linéaires en variables bivalentes", *Comptes Rendus, Académie des Sciences, Paris, 258*, 3817–3820.

Balas, E. (1964-2) "Extension de l'algorithme additif à la programmation en nombres entiers et la programmation nonlinéaire", *Comptes Rendus, Académie des Sciences, Paris, 258*, 5136–5139.

Balas, E. (1967) "Discrete Programming by the Filter Method", *Operations Research, 15*, 915–957.

Balas, E. (1968) "A Note on the Branch-and-Bound Principle", *Operations Research, 16*, 442–445.

Balas, E. (1970) "Duality in Discrete Programming", *Proc. Princeton Symp. on Math. Programming*, 179–198.

Balas, E. (1971) "The Intersection Cut — A New Cutting Plane for Integer Programming", *Operations Research, 19*, 19–39.

Balas, E. (1972-1) "Ranking the Facets of the Octahedron", *Discrete Mathematics, 2*, 1–15.

Balas, E. (1972-2) "Integer Programming and Convex Analysis: Intersection Cuts from Outer Polars", *Math. Progr., 2*, 330–382.

Balas, E. (1973) "A Note on the Group Theoretic Approach to Integer Programming and the 0–1 case", *Operations Research, 21*, 321–322.

Balas, E., Bowman, V. J., Glover, F. and Sommer, D. (1971) "An Intersection Cut from the Dual of the Unit Hypercube", *Operations Research, 19*, 40–44.

Balas, E. and Padberg, M. W. (1972) "On the Set-Covering Problem", *Operations Research, 20*, 1152–1161.

Balinski, M. L. (1961) "Fixed-Cost Transportation Problems", *Naval Research Logistic Quarterly, 8*, 41–54.

Balinski, M. L. (1967) "Some General Methods in Integer Programming", in *Nonlinear Programming*, Abadie ed. (1967).
Balinski, M. L. (1970-1) "Integer Programming: Methods, Uses, Computation", *Proc. Princeton Symp. Math. Programming*, 199–266.
Balinski, M. L. (1970-2) "On Recent Developments in Integer Programming", *Proc. Princeton Symp. Math. Programming*, 267–302.
Balinski, M. L. (1970-3) "On Maximum Matching, Minimum Covering and Their Connection", *Proc. Princeton Symp. Math. Programming*, 303–312.
Balinski, M. L. and Quart, R. E. (1964) "On an Integer Program for a Delivery Problem", *Operations Research, 12*, 300—304.
Balut, S. J. (1973) "Scheduling to Minimize the Number of Late Jobs When Setup and Processing Times are Uncertain", *Management Science, 19*, 1283–1289.
Baumol, W. J. and Kuhn, H. W. (1962) "An Approximate Algorithm for the Fixed-Charged Transportation Problem", *Naval Research Logistics Quarterly, 9*, 1–15.
Baumol, W. J. and Wolfe, P. (1958) "A Warehouse Location Problem", *Operations Research, 6*, 252–263.
Bauer, F. L. (1963) "Algorithm 153, Gomory", *Commun. Assoc. Computing Machinery 6*, No. 2.
Beale, E. M. L. (1963) "Two Transportation Problems", *Actes de la 3ième Confèrence Internationale de Recherche Operationelle*, Oslo, Paris 1964, 780–788.
Beale, E. M. L. (1965) "Survey of Integer Programming", *Operational Research Quarterly, 16*.
Beale, E. M. L. and Small, R. E. (1966) "Mixed Integer Programming by a Branch-and-Bound Technique", *Proc. IFIP Congress, 65, Vol. 2*. Ed. by W. H. Kalenich, 450–451.
Bellman, R. (1962) "Dynamic Programming Treatment of the Traveling Salesman Problem", *J. Assoc. Comp. Mach. 9*, 61–63.
Bellman, R. (1965) "Maximization over Discrete Sets", *Naval Research Logistics Quarterly. 3*, 67–70.
Bellman, R. E. (1957) *Dynamic Programming*, Princeton University Press, Princeton.
Bellman, R. E. and Dreyfus, S. E. (1962) *Applied Dynamic Programming*, Princeton University Press, Princeton.
Bellman, R. E. and Hall, M. Jr. eds (1960) *Combinatorial Analysis*, Proc. Tenth Symposium in Appl. Math. of Amer. Math. Soc.
Bellmore, M. and. Malone, J. C. (1971) "Pathology of Traveling Salesman Subtour Elimination Algorithms", *Operations Research, 19*, 278–307.
Bellmore, M. and Nemhauser, G. L. (1968) "The Traveling Salesman Problem: A Survey", *Operations Research, 16*, 538–558.
Bellmore, M. and Ratliff, H. D. (1971) "Set-Covering and Involuntary Bases", *Management Science, 18*, 184–206.
Benders, J. F. (1962) "Partitioning Procedures for Solving Mixed-Variable Programming Problems", *Numerische Mathematik, 4*, 238–252.
Benichou, M., Gauthier, J. M., Girodet, P. Heutges, G., Ribière G. and Vincent, O. (1971) "Experiments in Mixed Integer Programming", *Math. Programming, 1*, 76–94.
Ben-Israel, A. and Charnes, A. (1962) "On Some Problems of Diophantine Programming", Cahiers du Centre d'Études de Recherche Operationelle 4, 215–280.
Berge, C. (1957) "Two Theorems in Graph Theory", *Proc. National Academy of Sciences. 43*, 842–844.
Bertier, P., Nghiem Ph. T. and Roy, B. (1965) "Programmes Linéaires en Nombres Entiers", *METRA, 4*, No. 3.
Bertier, P. and Roy, B. (1965) "Une Procédure de Résolution pour une Classe de Problèmes Pouvant avoir un Caractère Combinatoire", *ICC Bulletin, 4*, 19–28.
Bessière, F. (1965) "Sur la Recherche du Nombre Chromatique d'un Graphe par un Programme Linéaire en Nombres Entiers", *Revue Française de Recherche Operationelle, 9*, 143–148.
Bowman, E. H. (1960) "Assembly Line Balancing by Linear Programming", *Operations Research, 8*, 385–389.

Bowman, V. J., Jr. and Nemhouser, G. L. (1970) "A Finiteness Proof for Modified Dantzig Cuts in Integer Programming", *Naval Research Logistics Quarterly*, *17*, 303–313.
Bowman, V. J. and Glover, F. (1972) "A Note on Zero–One Integer and Concave Programming", *Operations Research*, *20*, 182–183.
Bradley, G. H. (1969) "Equivalent Integer Programs", in *O. R. 69: Proc. 5th Int. Conf.* J. R. Lawrence (ed.), Tavistock Publications, London, 455–463.
Bradley, G. H. (1971–1) "Equivalent Integer Programs and Canonical Problems", *Management Science*, *17*, 354–366.
Bradley, G. H. (1971–2) "Algorithms for Hermite and Smith Normal Matrices and Linear Diophantine Equations", *Math. Comp.*, *25*, 897–908.
Bradley, G. H. (1971–3) "Transformation of Integer Programs to Knapsack Problems", *Discrete Mathematics*, *1*, 29–45.
Bradley, G. H. (1973) "Equivalent Mixed Integer Programming Problems", *Operations Research*, *21*, 323–325.
Bradley, G. H. and Wahi, P. N. (1973) "An Algorithm for Integer Linear Programming: A Combined Algebraic and Enumeration Approach", *Operations Research*, *21*, 45–60.
Brikker, V. I., (1971) "Ob odnoi Zadache Tselochislennogo Vypuklogo Programmirovaniya" (On a Problem in Integer Convex Programming) (Russian), Izvestiya Akademii Nauk, SSSR. *Tekhnicheskaya Kibernetika* (USSR) *3*, 48–53.
Burdet, C.-A. (1972) "Enumerative Inequalities in Integer Programming", *Math. Programming*, *2*, 32–64.
Burdet, C.-A. (1973) "Enumerative Cuts I", *Operations Research*, *21*, 61–89.

Cabot, A. V. and A. P. Hurter, (1968) "An Approach to Zero–One Integer Programming", *Operations Research*, *16*, 1206–1211.
Cabot, V. A. (1970) "A Bound-and-Scan Algorithm for the Knapsack Problem", *Operations Research*, *18*, 306–311.
Camion, P. (1960) "Une Méthode de Résolution par l'Algèbre de Boole des Problèmes Combinatoires ou Intervienment des Entiers", *Cahiers du Centre d'Études de Recherche Opérationelle*, *2*, 243–289.
Charnes, A. and W. W. Cooper, (1961), *Management Models and Industrial Applications of Linear Programming*, John Wiley, New York.
Cooper, L. and C. Drebes, (1967) "An Approximate Method for the Fixed Charge Problem", *Naval Research Logistics Quarterly*, *14*, 101–113.
Crowder, H. P. and E. L. Johnson (1973) "Use of Cyclic Group Methods in Branch-and-Bound", in *Mathematical Programming*, Hu and Robinson, eds., (1973), 213–226.

Dakin, R. J. (1965) "A Tree Search Algorithm for Mixed Integer Programming Problems", *The Computer Journal*, *8*, 250–255.
Dalton, R. E. and Llewellyn, R. W. (1967) "An Extension of the Gomory Mixed-Integer Algorithm to Mixed-Discrete Variables", *Management Science*, *12*, 569–575.
Dantzig, G. B. (1960) "On the Significance of Solving Linear Programming Problems with some Integer Variables", *Econometrica*, *28*, 30–44.
Dantzig, G. B. (1957) "Discrete-Variable Extremum Problems", *Operations Research*, *5*, 266–277.
Dantzig, G. B. (1959) "Note on Solving Linear Programs in Integers", *Naval Research Logistics Quarterly*, *6*, 75–76.
Dantzig, G. B. (1963) *Linear Programming and Extensions*, Princeton University Press, Princeton, N.J., USA.
Dantzig, G. B., Fulkerson, D. R. and Johnson, S. M. (1954) "Solution of a Large Scale Traveling Salesman Problem", *J. Op. Res. Soc. America*, *2*, 393–410.
Dantzig, G. B., Fulkerson, D. R. and Johnson, S. M. (1959) "On a Linear Programming, Combinatorial Approach to the Traveling Salesman Problem", *Operations Research*, *7*, 58–66.

Dantzig, G. B. and Ramsee, J. H. (1959) "The Truck Dispatching Problem", *Management Science, 6*, 80–91.
Davis, R. E., Kendrick, D. A. and Weitzman, M. (1971) "A Branch-and-Bound Algorithm for 0–1 Mixed Integer Programming Problems", *Operations Research, 19*, 1036–1044.
Delmas, L. and Henry-Labordère, A. (1971) "Solution Exacte du Problème d'Enlèvement des Modules", *METRA, 10*, 453–458.
De Mario, A., Montalenti, P. and Rodolfi, E. (1972) "Dimensionamento e Gestione Ottimale di una Rete di Transporti", *Ricerca Operativa, 2*, 27–44.
D'Esopo, D. A. and Lefkowitz, B. (1964) "Note on an Integer Linear Programming Model for Determining a Minimum Embarkation Fleet", *Naval Research Logistic Quarterly, 11*, 79–82.
D'Esopo, D. A. and B. Lefkowitz, (1963) "Certification of Algorithm 153", *Commun. Assoc. Comp. Mach., 6* No. 8.
Dinkelbach, W. (1965) On convex Integer Programming, *Colloquium on Applications of Mathematics to Economics*, Akadémiai Kiadó, Budapest, ed. A. Prékopa.
Driebeek, N. J. (1966) "An Algorithm for the Solution of Mixed Integer Programming Problems". *Management Science, 12*, 576–587.

Echols, R. E. and Cooper, L. (1968) "Solution of Integer Linear Programming Problems by Direct Search", *J. Assoc. Comp. Mach., 15*, 75–84.
Edmonds, J. (1962) "Covers and Packings in a Family of Sets", *Bull. Am. Math. Soc., 68*, 494–499.
Edmonds, J. (1965) "Maximum Matching and a Polyhedron with 0, 1-vertices", *J. Res. Nat. Bureau Standards, 69B*, 125–130.
Edmonds, J. (1965) "Paths, Trees and Flowers", *Can. J. Math. 17*, 449–467.
Edmonds, J. and Johnson, E. L. (1970) "Matching: A Well-Solved Class of Integer Linear Programs", *Proc. Calgary Int. Conf. Comb. Structs. and their Appl.*, 89–92, Gordon and Breach, New York.
Edmonds, J. and Karp, R. M. (1972) "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems", *J. Assoc. Comp. Mach., 19*, 248–264.
Efroymson, M. A. and Ray, T. L. (1966) "A Branch-Bound Algorithm for Plant Location", *Operations Research, 14*, 361–368.
Everett, III., Hugh, (1963) "Generalized Lagrange Multiplier Method for Solving Problems of Optimal Allocation of Resources", *Operations Research, 11*, 399–417.

Faaland, B. (1972) "On the number of Solutions to a Diophantine Equation", *J. Combinatorial Theory, 13*, 170–175.
Faaland, B. (1973) "Solution of Value-Independent Knapsack Problem by Partitioning", *Operations Research, 21*, 332–337.
Farkas, Gy. (1902) "Über die Theorie der einfachen Ungleichungen", *J. Reine Angew. Math. 124*, 1–24.
Feller, W. (1966) *An Introduction to Probability Theory and its Application, Vol. II.* John Wiley and Sons, New York.
Finkelstein, J. J. (1970) "Estimation of the Number of Iterations for Gomory's All-Integer Algorithm", *Dokl. Akad. Nauk. SSSR., 193*, 988–992.
Fleischmann, B. (1967) "Computational Experience with the Algorithm of Balas", *Operations Research, 15*, 153–155.
Florian, M. and Robillard, P. (1971) "Programmation Hyperbolique en Variables Bivalentes", *Revue Française d'Informatique et de Recherche Opérationelle, 5*, V–1, 3–9.
Ford, L. R. and Fulkerson, D. R. (1963) *Flows in Networks*, Princeton University Press, Princeton, USA.
Forgó, F. (1969) "Relationship between Mixed Zero–One Integer Linear Programming and Certain Quadratic Programming Problems", *Studia Scientiarum Mathematicarum Hungarica, 4*, 37–43.

Fortet, R. (1959) "L'Algébre de Boole et ses Applications en Recherche Opérationelle", *Cahiers du Centre d'Études de Recherche Opérationelle*, 1, No. 4. 5–36.
Freeman, R. F. (1966) "Computational Experience with a Balasian Integer Programming Algorithm", *Operations Research, 14*, 935–941.
Frehel, J. (1973) "Sur l'Utilisation de Troncatures de Gomory dans les Algorithmes Enumératifs", *Rev. Fran. Int. Inf. Rech. Opér., 7*, R–2, 5–16.
Fulkerson, D. R. (1971) "Blocking and Anti-Blocking Pairs of Polyhedra", *Math. Programming, 1*, 168–194.
Fulkerson, D. R. and Ryser, H. J. (1961) "Widths and Heights of (0, 1)-Matrices", *Can. J. Math., 13*, 239–255.

Gale, D. (1968) "Optimal Assignments in an Ordered Set: an Application of Matroid Theory", *J. Comb. Theory., 4*, 176–180.
Garfinkel, R. S. (1973) "On Partitioning the Feasible Set in a Branch-and-Bound Algorithm for the Asymmetric Traveling Salesman Problem", *Operations Research, 21*, 340–343.
Garfinkel, R. S. and Nemhouser, G. L. (1969) "The Set-Partioning Problem: Set-Covering with Equality Constraints", *Operations Research, 17*, 848–856.
Garfinkel, R. S. and Nemhouser, G. L. (1972) *Integer Programming*, John Wiley and Sons, New York.
Garfinkel, R. S. and Nemhouser, G. L. (1973) "A Survey of Integer Programming Emphasizing Computation and Relations Among Models", in *Mathematical Programming*, Hu and Robinson eds., (1973), 77–156.
Gavett, J. W. and Plyter, N. V. (1966) "The Optimal Assignment of Facilities to Locations by Branch and Bound", *Operations Research, 14*, 210–232.
Geoffffrion, A. M. (1967) "Integer Programming by Implicit Enumeration and Balas's Method", *SIAM Review, 9*, 178–190.
Geoffrion, A. M., ed. (1972), *Perspectives on Optimization: A Collection of Expository Articles*, Addison-Wesley, Reading, Massachusetts.
Geoffffrion, A. M. and Marsten, R. E. (1972) "Integer Programming: A Framework and State-of-the-Art Survey", *Management Science, 18*, 465–491.
Giglio, R. J. and Wagner, Harvey M. (1966) "Approximate Solutions to the Three Machine Scheduling Problem", *Operations Research, 12*.
Gilmore, P. C. (1962) "Optimal and Suboptimal Algorithms for the Quadratic Assignment Problem", *J. Soc. Indust. & Appl. Math., 10*, 305–313.
Gilmore, P. C. and Gomory, R. E. (1961), (1963) "A Linear Programming Approach to the Cutting Stock Problem", Part I: *Operations Research, 9*, 849–859; "A Linear Programming Approach to the Cutting Stock Problem", Part II: *Operations Research, 11*, 863–888.
Gilmore, P. C. and Gomory, R. E. (1964) "A Solvable Case of the Traveling Salesman Problem", *Proc. Nat. Academy Sciences, 51*, 178–181.
Gilmore, P. C. and Gomory, R. E. (1965) "Multi Stage Cutting Stock Problems of Two and More Dimensions", *Operations Research, 13*, 94–120.
Gilmore, P. C. and Gomory, R. E. (1966) "The Theory and Computation of Knapsack Functions", *Operations Research, 14*, 1045–1074.
Gleason, A. M. (1960) "A Search Problem in the n-Cube" in *Combinatorial Analysis*, Proc. Tenth Symposium in Appl. Math. of Am. Math. Soc. Bellman and Hall, eds (1960), 175–178.
Glover, F. (1965–1) "A Bound Escalation Method for the Solution of Integer Linear Programs", Cahiers du Centre d'Études de Recherche Opérationelle 7, 131–168.
Glover, F. (1965–2) "A Hybrid-Dual Integer Programming Algorithm" Cahiers du Centre d'Études de Recherche Opérationelle 7, 5–23.
Glover, F. (1965–3) "A Multiphase-Dual Algorithm for the Zero–One Integer Programming Problem", *Operations Research, 14*, 879–919.
Glover, F. (1967) "Stronger Cuts in Integer Programming", *Operations Research, 15*, 1174–1177.

Glover, F. (1968–1) "A New Foundation for a Simplified Integer Programming Algorithm", *Operations Research, 16*, 727–740.
Glover, F. (1968–2) "Surrogate Constraints", *Operations Research, 16*, 741–749.
Glover, F. (1972) "Cut-Search Methods in Integer Programming", *Math. Prog., 3*, 86–100.
Glover, F. (1973–1) "Convexity Cuts and Cuts Search", *Operations Research, 21*, 123–134.
Glover, F. (1973–2) "Convexity Cuts for Multiple Choice Problems", *Discrete Mathematics, 6*, 221–234.
Glover, F. and Karney D. and Klingman, D. (1972) "The Augmented Predecessor Index Method for Locating Stepping-Stone Paths and Assigning Dual Prices in Distribution Problems", *Trans. Sci., 2*, 171–179.
Glover, F. and Klingman, D. (1973–1) "Concave Programming Applied to a Special Class of 0–1 Integer Programs", *Operations Research, 21*, 135–140.
Glover, F. and Klingman, D. (1973–2) "The Generalized Lattice-Point Problem", *Operations Research, 21*, 141–155.
Glover, F. and Woolsey, E. (1973) "Further Reduction of Zero-One Polynomial Programming Problems to Zero-One Linear Programming Problems," *Operations Research, 21*, 156–161.
Glover, F. and Zionts, S. (1965) "A Note on the Additive Algorithm of Balas", *Operations Research, 13*, 546–549.
Golomb, S. W. and Baumert, L. D. (1965) "Backtrack Programming", *J. Assoc. Comp. Mach., 12*, 516–524.
Gomory, R. E. (1958) "Outline of an Algorithm for Integer Solutions to Linear Programs" *Bull. Am. Math. Soc., 64*, 275–278.
Gomory, R. E. (1960) "All-Integer Programming Algorithm", in *Industrial Scheduling*, Muth and Thompson eds., (1963), 193–206. (First issued as IBM Research Report RC–189)
Gomory, R. E. (1963–1) "Large and Non-Convex Problems in Linear Programming", in *Experimental Arithmetic, High Speed Computing and Mathematics*, Proc. Symp. Appl. Math. 14, American Mathematical Society.
Gomory, R. E. (1963–2) "An Algorithm for Integer Solutions to Linear Programs", in *Recent Advances in Mathematical Programming*, Graves and Wolfe eds., (1963), (editors), 269–302.
Gomory, R. E. (1965) "On the Relation Between Integer and Non-Integer Solutions to Linear Programs", *Proc. Nat. Acad. Sciences, 53*, 260–265.
Gomory, R. E. (1967) "Faces of an Integer Polyhedron", *Proc. Nat. Acad. Sciences, 57*, 16–18.
Gomory, R. E. (1968) "Some Polyhedra Connected with Combinatorial Problems", *Lin. Alg. Appl., 2*, 451–558.
Gomory, R. E. and Baumol, W. J. (1960) "Integer Programming and Pricing", *Econometrica, 28*, 521–550.
Gomory, R. E. and Hoffman, A. J. (1963) "On the Convergence of an Integer Programming Process", *Naval Research Logistics Quarterly, 10*, 121–123.
Gomory, R. E. and Johnson, E. L. (1972) "Some Continuous Functions Related to Corner Polyhedra", *Math. Programming, 3*, 23–85.
Gomory, R. E. and Johnson, E. L. (1973) "The Group Problems and Subadditive Functions", in *Mathematical Programming*, Hu and Robinson, eds (1973), 157–184.
Gondran, M. (1973) "Un Outil de la Programmation en Nombres Entiers: 'La Méthode des Congruences Décroissantes'", *Rev. Fran. Aut. Inf. Rech. Opér., 7*, V-3, 35–54.
Gorry, G. A., Shapiro, J. F. and Wolsey, L. A. (1972) "Relaxation Methods for Pure and Mixed Integer Programming Problems", *Management Science, 18*, 229–239.
Gorry, G. A. and Shapiro, J. F. (1971) "An Adaptive Group Theoretic Algorithm for Integer Programming Problems", *Management Science, 17*, 285–306.
Graves, G. and Whinston, A. (1968) "A New Approach to Discrete Mathematical Programming", *Management Science, 15*, 177–189.
Graves, R. L. and Wolfe, P. (eds), (1963) *Recent Advances in Mathematical Programming*, McGraw-Hill, New York.

Greenberg, H. H. (1968) "A Branch-Bound Solution to the General Scheduling Problem", *Operations Research, 16*, 353–361.
Greenberg, H. (1969) "A Dynamic Programming Solution to Integer Linear Programs", *J. Math. Analysis and Applications, 26*, 454–459.
Greenberg, H. (1971) *Integer Programming*, Academic Press, New York.
Greenberg, H. and Hegenich, R. L. (1970) "A Branch Search Algorithm for the Knapsack Problem", *Management Science, 16*, 327–332.
Guha, D. K. (1973) "The Set-Covering Problem with Equality Constraints", *Operations Research, 21*, 348–351.
Gupta, H. (1970) "Partitions – A Survey", *J. Nat. Bureau Standards-B, 74B*, 1–29.

Hadley, G., (1962) *Linear Programming*, Addison—Wesley, Reading, Massachusetts.
Hadley, G. (1964) *Nonlinear and Dynamic Programming*, Addison—Wesley, Reading, Massachusetts.
Haldi, J. (1964) "25 Integer Programming Test Problems", *Working Paper No. 43*, Stanford University, Stanford, USA.
Haldi, J. and Isaacson, L. M. (1965) "A Computer Code for Integer Solutions to Linear Programs", *Operations Research, 13*, 946–968.
Hall, M. and Knuth, D. E. (1955) "Combinatorial Analysis and Computers", Herbert Ellswort Slaught Memorial Papers, No. 10, Supplement, *Amer. Math. Monthly 72*, 21–28.
Hammer, P. L. and Rudeanu, S. (1968), *Boolean Methods in Operations Research and Related Areas*, Springer, New York.
Harris, P. M. J. (1964) "The Solution of Mixed Integer Linear Programs", *Operational Research Quarterly, 15*, 117–135.
Healy, W. C., Jr. (1964) "Multiple Choice Programming", *Operations Research, 12*, 112–138.
Held, M. and Karp, R. M. (1962) "A Dynamic Programming Approach to Sequencing Problems", *J. Soc. Indust. & Appl. Math., 10*, 196–210.
Held., M. and Karp, R. M. (1970) "The Traveling-Salesman Problem and Minimum Spanning Trees", *Operations Research, 18*, 1138–1162.
Held, M. and Karp, R. M. (1971) "The Traveling-Salesman Problem and Minimum Spanning Trees: Part II", *Math. Programming, 1*, 6–25.
Held, M. and Karp, M. R. and Shareshian, R. (1963) "Assembly-Line Balancing by Dynamic Programming with Precedence Constraints", *Operations Research, 11*, 442–459.
Hess, S. W., Siegfeldt, H. J., Weaver, J. B., Whelan J. N. and Zitlaw, P. A. (1965) "Nonpartisan Political Redistricting by Computer", *Operations Research, 13*, 998–1006.
Hillier, F. S., (1967) "Chance-Constrained Programming with 0–1 or Bounded Continuous Decision Variables", *Management Science, 14*, 34–57.
Hillier, F. S. (1969–1) "Efficient Heuristic Procedures for Integer Linear Programming with an Interior", *Operations Research, 17*, 600–637.
Hillier, F. S. (1969–2) "A Bound-and-Scan Algorithm for Pure Integer Linear Programming with General Variables", *Operations Research, 17*, 638–679.
Hirsch, W. M. and Dantzig, G. B. (1968) "The Fixed Charge Problem", *Naval Research Logistics Quarterly, 15*, 413–424.
Hoffman, A. J. (1960) "Some Recent Applications of the Theory of Linear Inequalities to Extremal Combinatorial Analysis", in *Combinatorial Analysis*, Bellman and Hall, eds (1960), 113–128.
Hofstedt, K. and Thamelt, W. (1971) "Über einen Algorithmus zur Lösung Gemischt-Ganzzahliger Optimalprobleme", *Mathematische Operationsforschung und Statistik, 2*, 199–222.
House, R. W., Nelson, L. D. and Rado, T. (1965) "Computer Studies of a Certain Class of Linear Integer Problems", in *Recent Advances in Optimization Techniques*, Lavi and Vogl eds. (1965), 241–280.
Hu, T. C. (1969) *Integer Programming and Network Flows*, Addison–Wesley, Reading, Massachusetts.
Hu, T. C. (1970) "On the Asymptotic Integer Algorithm", *Lin. Alg. Appl., 3*, 279–294.

Hu, T. C. (1971) "Some Problems in Discrete Optimization", *Mathematical Programming, 1*, 102–112.
Hu, T. C. and Robinson, S. M. eds (1973) *Mathematical Programming*, Academic Press, New York.
Huard, P. (1967) "Resolution of Mathematical Programming with Nonlinear Constraints by the Method of Centres", in *Nonlinear Programming* Abadie ed. (1967).
Huard, P. (1970) "Programmes Mathématiques Nonlineares en Variables Bivalentes", *Proc. Princeton Symp. on Math. Programming*, 313–322.

IBM, (1964) Data Processing Division, *Proc. IBM Scientific Computing Symp. on Combinatorial Problems*, White Plains, New York.
Ignall, E. and Schrage L., (1965) "Application of the Branch-and-Bound Technique to Some Flow-Shop Scheduling Problems", *Operations Research, 13*, 400–412.
Ignatenko, P. P. (1971) "K Resheniyu Transportnoi Zadachi s Fiksirovannymi Doplatami" (On the Solution of the Transportation Problem with Fixed Payments) (Russian), *Kibernetika, 1*, 146–147.
Ivanescu, P. L. and Rudeanu, S. (1966) "Pseudo-Boolean, Methods for Bivalent Programming. *Lecture Notes in Mathematics*", Springer, Berlin, New York, 120.

Jensen, P. A. (1971) "Optimum Network Partitioning", *Operations Research, 19*, 916–932.
Jeroslow, R. G. (1971) "Comments on Integer Hulls of Two Linear Constraints", *Operations Research, 19*, 1061–1069.
Jeroslow, R. G. and Kortanek, K. O. (1971) "On an Algorithm of Gomory", *J. SIAM, 21*, 55–60.
Johnson, E. L. (1973) "Cyclic Groups, Cutting Planes and Shortest Path", in *Mathematical Programming*, Hu and Robinson eds., (1973), 185–212.
Joseph, J. A. (1967) "Heuristic Approach to Nonstandard Form Assignement Problems", *Operations Research, 15*, 680–693.

Kalymon, B. A. (1971) "Note Regarding, A New Approach to Discrete Mathematical Programming", *Management Science, 17*, 777–778.
Kaplan, Seymour (1966) "Solution of the Lorie-Savage and Similar Integer Programming Problems by the Generalized Lagrange Multiplier Method", *Operations Research, 14*, 1130–1136.
Karp, R. M. (1961) "Minimum-Redundancy Coding for the Discrete Noiseless Channel", *IRE Trans. Prof. Group on Inf. Theory*, IT–7, No. 1., 27–38.
Karp, R. M. and Held, M. (1967) "Finite-State Processes and Dynamic Programming", *SIAM J. Appl. Math., 15*, 693–718.
Karp, R. M., McFarlin, F. R., Roth, J. P. and Wills, J. R. (1961) "A Computer Program for the Synthesis of Combinatorial Switching Circuits", *Proc. Second Symp. of Switching Institute of Electrical Engineers, Publ. S–143.*
Kelley, J. E., Jr. (1960) "The Cutting Plane Method for Solving Convex Programs", *J. SIAM, 8.*
Kianfar, F. (1971) "Stronger Inequalities for 0–1 Integer Programming Using Knapsack Functions", *Operations Research, 19*, 1374–1392.
Klafszky, E. (1970), "Hálózati folyamok" (Network Flows) (Hungarian), *Lecture Notes: Mathematical Methods of O.R.*, 1967–69, ed. A. Prékopa, János Bolyai Mathematical Society, Budapest.
Klein, M. and Takamori H., (1972) "Parallel Line Assignment Problems", *Management Science, 19*, 1–11.
Kolesar, P. J. (1967) "A Branch-and-Bound Algorithm for the Knapsack Problem", *Management Science, 13*, 723–735.
Koopmans, T. C. and Beckman, M. (1957) "Assignment problems and the Location of Economic Activities", *Econometria, 25*, 53–76.

Kovács, Á. and Stahl, J. (1971) "A vállalati beruházási politika optimálásának egy modellje (A Model for the Optimization of Investment Policy in a Firm)" (Hungarian, English and Russian summaries), *Szigma*, *4*, 59–67.

Kovács, L. B. (1968) "Leszámlálási struktúrák és alkalmazásuk diszkrét programozási feladatok megoldására" (Enumeration Structures and Their Application for Solving Discrete Programming Problems) (Hungarian), *Matematikai Lapok*, *XIX*, 33–48.

Kovács, L. B. (1969), "A diszkrét programozás kombinatorikus módszerei" (Combinatorial Methods of Discrete Programming) (Hungarian) *Lecture Notes: Mathematical Methods of O.R.*, ed. A. Prékopa, János Bolyai Mathematical Society, Budapest.

Kovács, L. B. (1973) "A New Solution for the General Set Covering Problem", 5th Conference on Optimization Techniques, Part I, *Lecture Notes in Computer Science*, *3*, Springer, 471–483.

Kovács, L. B. and Nagykálnai, E. (1968) "Korlátozott hozzárendelési probléma és mezőgazdasági alkalmazása". (Restricted Assignment Problem and its Application in Agriculture, Hungarian) *MTA III. Osztály Közleményei*, *18*, 3–21.

Krolak, P. D. (1969) "Computational Results of an Integer Programming Algorithm", *Operations Research*, *17*, 743–748.

Kraft, D. H. and Hill, T. W. Jr. (1973) "The Journal Selection Problem in a University Library System", *Management Science*, *19*, 613–626.

Krawczyk, R., (1971) "Die Anwendung der Methode 'Branch-and-Bound' auf ein Verallgemeinertes Knapsackproblem", *Angewandte Informatik*, *13*, 461–468.

Kuehn, A. A. and Hamburger, M. J. (1963) "A Heuristic Program for Locating Warehouses", *Management Science*, *9*, 643–666.

Kuhn, H. W. and Tucker, A. W. (1956) (eds), *Linear Inequalities and Related Systems*, Annals of Mathematics Study No. 38, Princeton University Press, Princeton, N.T., USA.

Kunzi, H. P. and Oettli, Werner (1963) "Integer Quadratic Programming", in *Recent Advances in Mathematical Programming*, Graves and Wolfe, eds (1963) 303–308.

Lambert, F. (1960) "Programmes Linéaires Mixtes", *Cahiers du Centre d'Études de Recherche Opérationelle*, *2*, 47–75 and 75–126.

Land, A. H. and Doig, A. G. (1960) "An Automatic Method of Solving Discrete Programming Problems", *Econometrica*, *28*, 497–520.

Land, A. H. and Doig, A. (1965) "A Problem of Assignment with Interrelated Costs", *Operational Research Quarterly*, *14*, 185–199.

Lasdon, L. S. (1971) "An Efficient Algorithm for Multi-Item Scheduling", *Operations Research*, *19*, 946–969.

Lavi, A. and Vogl, T. P. (1965), (eds), *Recent Advances in Optimization Techniques*, John Wiley and Sons, New York .

Lawler, E. L. (1963) "The Quadratic Assignment Problem", *Management Science*, *9*, 586–599.

Lawler, E. L. (1966) "Covering Problems: Duality Relations and a New Method of Solution", *SIAM J. Appl. Math.*, *14*, 1115–1132.

Lawler, E. L. (1972) "A Procedure for Computing the *K* Best Solutions to Discrete Optimization Problems and its Application to the Shortest Path Problem", *Management Science*, *18*, 401–405.

Lawler, E. L. and Bell, M. D. (1966) "A Method for Solving Discrete Optimization Problems", *Operations Research*, *14*, 1098–1112.

Lawler, E. L. and Wood, D. E. (1966) "Branch-and-Bound Methods. A Survey", *Operations Research*, 13, 699–719.

Lemke, C. E., Salkin, H. M. and Spielberg, K. (1971) "Set Covering by Single Branch Enumeration with Linear Programming Subproblems", *Operations Research*, *19*, 998–1022.

Lemke, C. E. and Spielberg, K. (1967) "Direct Search Algorithms for the Zero-One and Mixed Integer Programming", *Operations Research*, *15*, 892–914.

Lene, O. (1972) "Methoden zur Lösung Dreidimensionaler Zuordnungsprobleme", *Angewandte Informatik*, *14*, 154–162.

Lin, Shen, (1965) "Computer Solutions of the Traveling Salesman Problem", *Bell System Technical Journal, XLIV*, 2245–2269.
Lippman, S. A. (1969) "Planning Horizon Theorems for Knapsack and Renewal with a Denumerable Number of Activities", *Operations Research, 17*, 163–174.
Little, J. D. C. (1966) "The Synchronization of Traffic Signals by Mixed Integer Linear Programming", *Operations Research, 14*, 568–594.
Little, J. D. C., Murty, K. G., Sweeney, D. W. and Karel, C. (1963) "An Algorithm for the Traveling Salesman Problem", *Operations Research, 11*, 972–989.
Loehman, E., Ngiem, Ph. T. and Whinston, A. (1970) "Two Algorithms for Integer Optimization", *Revue Informatique et Recherche Opérationelle, 4*, V–2, 43–63.
Lomniczki, Z. A. (1965) "A Branch-and-Bound Algorithm for the Exact Solution of the Three-Machine Scheduling Problem," *Operational Research Quarterly, 16*, 89–100.

Manne, A. S. (1960) "On the Job-shop Scheduling Problem", *Operations Research, 8*, 219–223.
Manne, A. S. (1964) "Plant Location under Economies of Scale-Decentralization and Computation", *Management Science, 11*, 213–235.
Markowitz, H. M. and Manne, A. S. (1957) "On the Solution of Discrete Programming Problems", *Econometria, 25*, 84–110.
Martin, G. T. (1963) "An Accelerated Euclidean Algorithm for Integer Linear Programming", in *Recent Advances in Mathematical Programming*, Graves and Wolfe, eds. (1963).
Mao, J. C. T. and Wallingford, B. A. (1968) "An Extension of Lawler and Bell's Method of Discrete Optimization with Examples from Capital Budgeting", *Management Science, 15*, B51–B60.
Mathews, G. B. (1897) "On the Partition of Numbers", *Proc. London Math, Soc. 28*, 486–490.
McCluskey, E. J. (1956) "Minimization of Boolean Functions", *Bell System Technical Journal, 35*, 1417–1444.
McCluskey, E. J. (1959) "Error Correcting Codes — A Linear Programming Approach", *Bell System Technical Journal, XXXVIII*, 1485–1512.
McMahon, G. B. and Burton, P. G. (1967) "Flow-Shop Scheduling with the Branch-and-Bound Method", *Operations Research, 15*, 473–481.
Mensch, G., (1971) "Single Stage Linear Programming Zero–One Solutions to Some Traveling Salesman Type Problems", *INFOR, 9*, 282–292.
Miller, C. E., Tucker, A. W. and Zemlin, R. A. (1969) "Integer Programming Formulation of Traveling Salesman Problems", *J. Assoc. Comp. Mach., 7*, 326–329.
Minkowski, H. (1896) *Geometrie der Zahlen*, B. G. Treubner, Leipzig and Berlin, 1910. 256 pp.; reprinted by Chelsea Publishing Co., New York, 1953.
Mitten, L. G. (1970) "Branch-and-Bound Methods: General Formulation and Properties", *Operations Research, 18*, 24–34. See also a correction in *Operations Research 19*, 550 (1971).
Mizukami, K. (1968) "Optimum Redundancy for Maximum System Reliability by the Method of Convex and Integer Programming", *Operations Research, 16*, 392–406.
Morlet, E. (1960) "Codage de la Résolution des Programmes Linéaires Mixtes", Cahiers du Centre d'Études de Recherche Operationelle, *2*, 127–160.
Mueller-Merbach, H., (1970) *Approximation Methods for Integer Programming*, Johannes Gutenberg University, Mainz, FGR.
Murthy, K. G. (1968–1) "Solving the Fixed Charge Problem by Ranking the Extreme Points", *Operations Research, 16*, 268–279.
Murthy, K. G. (1968–2) "An Algorithm for Ranking All the Assignments in Order of Increasing Cost.", *Operations Research, 16*, 682–687.
Murthy, K. G. (1972) "A Fundamental Problem in Linear Inequalities with Applications to the Traveling Salesman Problem", *Math. Prog., 2*, 296–308.
Muth, J. F. and Tompson, G. L. eds (1963), *Industrial Scheduling*, Prentice-Hall, New York.

Nanda, P. S. (1973) "The Equivalence of Integer Programms to Constrained Recursive Systems", *Management Science, 19*, 809–824.
Nemhauser, G. L. (1966) *Introduction to Dynamic Programming* ,Wiley, New York.
Nemhauser, G. L. and Ulman, Z. (1968) "A Note on the Generalized Lagrange Multiplier Solution to an Integer Programming Problem", *Operations Research, 16*, 450–452.
Norman, R. Z. and Rabin, M. O., (1959) "An Algorithm for the Minimum Cover on a Graph", *Proc. Am. Math. Soc., 10*, 315–319.

Ouyahia, Ait, (1962) "Programmes Lineares a Variables Discretes", *Revue Française de Recherche Opérationelle, 6*, 55–75.

Palmer, D. S. (1965) "Sequencing Jobs Through a Multi-Stage Process in the Minimal Total Time", *Operations Research Quarterly, 16*, 101–107.
Pandit, S. N. Narahari, (1962) "The Loading Problem", *Operations Research, 10*, 639–646.
Petersen, C. C. (1967) "Computational Experience with Variants of the Balas Algorithm Applied to the Selection of R and D projects", *Management Science, 13*, 736–750.
Petersen, J. (1891) "Die Theorie der Regulären Graphen", *Acta Mathematica, 51*, 193–220.
Picard, J. C. and Ratliff, H. D. (1973) "A Graph-Theoretic Equivalence for Integer Programs", Operations *Research, 21*, 261–269.
Pierce, J. F. and Lasky, J. S. (1973) "Improved Combinatorial Programming Algorithms for a Class of All-Zero–One Integer Programming Problems", *Management Science, 19*, 528–543.
Pinkus, Ch. E., Gross, D. and Soland, R. M. (1973) "Optimal Design of Multiactivity Multifacility Systems by Branch-and-Bound", *Operations Research, 21*, 270–283.
Plane, D. R. and McMillan, C. Jr. (1971) *Discrete Optimization: Integer Programming and Network Analysis for Management Decisions*, Prentice-Hall, New York.
Prékopa, A. (1968) "Lineáris Programozás" (Linear Programming) (Hungarian), *Lecture Notes: Mathematical Methods of O.R.*, 1967–69, ed. A. Prékopa, János Bolyai Mathematical Society, Budapest.
Prékopa, A. *Linear Programming* Akadémiai Kiadó, Budapest, (in press).
Pyne, I. B. and McCluskey, E. J., Jr. (1961) "An Essay on Prime Implicant Tables", *J. Soc. Indust. and Appl. Mathematics, 9*, 604–631.

Quine, W. V. (1955) "A Way to Simplify Truth Functions", *Amer. Math. Monthly, 62*, 627–631.

Rado, F. (1963) "Linear Programming with Logic Conditions", *Communicarile Academie Republici Populare Romine, 13*, 1039–1041.
Rebelein, P. R. (1968) "An Extension of the Algorithm of Dirichlet for Solving Mixed Integer Programming Problems", *Operations Research, 16*, 193–198.
Reiter, S. and Rice, D. B. (1966) "Discrete Optimizing Solution Procedures for Linear and Nonlinear Integer Programming Problems", *Management Science, 12*, 829.
Reiter, S. and Sherman, G. (1965) "Discrete Optimizing", *J. Soc. Indust. and Appl. Mathematics, 13*, 864–889.
Raghavachari, M. (1969) "On Connections between Zero–One Onteger Programming and Concave Programming Under Linear Constraints", *Operations Research, 17*, 680–684.
Roth, J. P. (1958) "Algebraic Topological Methods for the Synthesis of Switching Systems, I", *Trans. Am. Math. Soc., 88*, 301–326.
Roth, J. P. and Wagner, E. G. (1960) "Minimization over Boolean Trees", *IBM J. of Research, 1*, 543–558.
Roth, R. (1969) "Computer Solutions to Minimum-Cover Problems", *Operations Research, 17*, 455–465.
Roth, R. H. (1970) "An Approach to Solving Linear Discrete Optimization Problems", *J. Assoc. Comp. Mach., 17*, 303–313.

Roth, J. P. and Karp, R. M. (1962) "Minimization over Boolean Graphs", *IBM J. of Research, 6*, 227–238.
Roth, J. P. and Wagner, E. G. (1959) "Algebraic Topological Methods for the Synthesis of Switching Systems, Part III: Minimization of Nonsingular Boolean Trees", *IBM J. of Research, 3*, 326–344.
Roy, B., Benayoun R. and Tergny, J. (1970) "From S.E.P. Procedure to the Mixed Ophelie Program", in *Nonlinear Programming*, Abadie ed., (1970) 419–436.
Rubin, D. S. (1970) "On the Unlimited Number of Faces in Integer Hulls of Linear Programs with a Single Constraint", *Operations Research, 18*, 940–946.
Rubin, J. (1973) "A Technique for the Solution of Massive Set Covering Problems, with Application to Airline Crew Scheduling", *Trans. Sci., 7*, 34–48.
Rutledge, R. W. (1967) "A Simplex Method for Zero–One Mixed Integer Linear Programs", *J. Math. Anal. and Appl., 18*, 377–390.

Saaty, T. L. (1970) *Optimization in Integers and Related Extremal Problems*, McGraw Hill, New York.
Salkin, H. M. (1970) "On the Merit of the Generalized Origin and Restart in Implicit Enumeration", *Operations Research, 18*, 549–554.
Senju, S. and Toyoda, Y. (1968) "An Approach to Linear Programming with 0–1 Variables", *Management Science, 15*, B196–B207.
Shapiro, J. F. (1968–1) "Dynamic Programming Algorithms for the Integer Programming Problem I: The Integer Programming viewed as a Knapsack Type Problem", *Operations Research, 16*, 103–121.
Shapiro, J. F. (1968–2) "Group Theoretic Algorithms for the Integer Programming Problem II. Extensions to a General Algorithm", *Operations Research, 16*, 928–947.
Shapiro, J. F. (1968–3) "Shortest Route Methods for Finite State Space Deterministic Dynamic Programming Problems", *J. SIAM, 16*, 1232–1250.
Shapiro, J. F. and Wagner, H. M. (1967) "A Finite Renewal Algorithm for the Knapsack Problem and Turnpike Models", *Operation Research, 15*, 319–341.
Shier, D. R. (1973) "A Decomposition Algorithm for Optimality Problems in Tree-Structured Networks", *Discrete Mathematics, 9*, 175–189.
Smith, H. J. S. (1861) "On Systems of Linear Indeterminate Equations and Congruences", *Phil. Trans, 151*, 293–326.
Spielberg, K. (1969–1) "Algorithms for the Simple Plant-Location Problem with Some Side Conditions", *Operations Research, 17*, 85–111.
Spielberg, K. (1969–2) "Plant Location with Generalized Search Origin", *Management Science, 16*, 165–178.
Srinivasan, A. V. (1965) "An Investigation of Some Computational Aspects of Integer Programming", *J. Assoc. Comp. Mach., 12*, 525–535.
Stanley, E. D., Honig D. P. and Gainen, L. (1954) "Linear Programming in Bid Evaluation", *Naval Research Logistic Quarterly, 1*, 48–54.
Steinberg, D. I. (1970) "The Fixed-Charge Problem", *Naval Research Logistics Quarterly, 17*, 217–236.
Story, A. E. and Wagner, H. M. (1963) "Computational Experience with Integer Programming for Job-Shop Scheduling", in *Industrial Scheduling*, Muth and Thompson, eds (1963).
Svestka, J. A. and Huckfeldt, V. E. (1973) „Computational Experience with an *M*-Salesman Traveling Salesman Algorithm", *Management Science, 19*, 790–799.

Taha, H. A. (1972) "Further Improvements in the Polynomial Zero–One Algorithm", *Management Science, 19*, B106–B107.
Thangavelu, S. R. and Shetty, C. M. (1971) "Assembly Line Balancing by 0–1 Integer Programming", *AIIE Trans. III*, 64–69.
Thiriez, H. (1971) "The Set Covering Problem: A Group Theoretic Approach." *Revue Française d'Informatique et de Recherche Opérationelle, 5*, 83–103.

Thompson, G. L. (1964) "The Stopped Simplex Method, Parts I, II", *Revue Francaise de Recherche Opérationelle, 8*, 159–182; *Revue Française de Recherche Opérationelle, 9*.
Tomlin, J. A. (1970) "Branch-and-Bound Methods for Integer and Non-Convex Programming", in *Nonlinear Programming*, Abadie ed., (1970), 437–450.
Tomlin, J. A. (1971) "An Improved Branch-and-Bound Method for Integer Programming", *Operations Research, 19*, 1070–1074.
Toregas, C., Swain, R., Revelle, C. and Bergman, L. (1971) "The Location of Emergency Service Facilities", *Operations Research, 19*, 1363–1373.
Trauth, C. A., Jr. and Woolsey, R. E. (1969) "Integer Linear Programming: A Study in Computational Efficiency", *Management Science, 15*, 481–493.
Tui, Hoang (1964) "Concave Programming under Linear Constraints", (Russian), *Doklady Akademii Nauk SSSR*, 1964; English Translation in *Soviet Mathematics* (1964), 1437–1440.

Veinott, A. F., Jr. and Dantzig, G. N. (1968) "Integral Extreme Points", *SIAM Rev., 10*, 371–372.
Verebriusova, A. (1904) "On the Number of Solutions of Indefinite Equations of the First Degree with Many Variables", *Mathematicheskii Sbornik, 24*, 662–688.
Votyakov, A. A. (1972) "Tselochislennoe Programmirovanie, Sravnenie Otsechenii" (Integer Programming Comparison of Cuts), (Russian) *Ekonomica i Matematicheskie Metody, 8*, 107–116.

Wagner, H. M. (1959) "An Integer Linear Programming Model for Machine Shop Scheduling", *Naval Research Logistics Quarterly, 6*, 131–140.
Wahi, P. N. and Bradley, G. H. (1961) "Integer Programming Test Problems", *Technical Report No. 28*, Yale University, Conn., USA.
Walker, R. J. (1960) "An Enumerative Technique for a Class of Combinatorial Problems", in *Combinatorial Analysis*, Bellman and Marschall, eds. (1960).
Warner, D. M. and Prawda, J. (1972) "A Mathematical Programming Model for Scheduling Nursing Personnel in a Hospital", *Management Science, 19*, 411–422.
Watters, L. J. (1967) "Reduction of Integer Polinomial Programming Problems to Zero–One Linear Programming Problem", *Operations Research, 15*, 1171–1174.
Weingartner, H. M. and Ness ,D. N. (1967) "Methods for the Solution of Multi-Dimensional 0–1 Knapsack Problems", *Operations Research, 15*, 83–103.
Wilson, R. S. (1967) "Stronger Cuts in Gomory's All-Integer Programming Algorithm", *Operations Research, 15*, 155–157.
Witzgall, C. (1963) "An All-Integer Programming Algorithm with Parabolic Constraints", *J. Soc. Indust. and Appl. Math., 11*, 855–871.
Wolsey, L. A. (1971–1) "Group Theoretic Results in Mixed Integer Programming", *Operations Research, 19*, 1691–1697.
Wolsey, L. A. (1971–2) "Extensions of the Group Theoretic Approach in Integer Programming", *Management Science, 18*, 74–83.
Woolsey, R. E. and Trauth, C. A. Jr. (1966) IPSC, "A Machine Independent Integer Linear Program", *Sandia Corporation Report SC-RR-66-433.*

Yen, J. Y. (1971) "On Hu's Decomposition Algorithm for Shortest Paths in a Network", *Operations Research, 19*, 983–985.
Young, R. D. (1968) "A Simplified (All-Integer) Integer Programming Algorithm", *Operations Research, 16*, 750–782.
Young, R. D. (1971)" Hypercylindrically Deduced Cuts in Zero–One Integer Programming", *Operations Research, 19*, 1393–1405.

Zak, Yu. A. (1972) "Algoritmy Resheniya Zadach, n Kommivogazherov' (Solution Algorithms for n-Travelling Salesman Problems')" (Russian with English summary), *Kibernetika, 1*, 99–106.
Zangwill, W. I., (1965) "Media Selection by Decision Programming", *J. Advert. Res., 5*, 23–27.

CHAPTER 12

RECENT DIRECTIONS IN DISCRETE PROGRAMMING

The aim of the present chapter is to give a wide view of new results and new directions in integer programming. For this purpose a considerable number of papers and books were selected and referred to mainly from the last 3–4 years. A classified bibliography, Kasting (1976) containing several thousand items, covers the previous time.

My intention has not been to provide a full list of publications but rather a representative sample with more papers on popular subjects. Priority was given to papers of survey nature with useful lists of further references in order to make the presentation compact, and papers with numerical experimentation. Personal comments were added especially to Sections 12.2 and 12.3 on general and special algorithms respectively.

12.1 Basic research supporting algorithms of discrete programming

12.1.1 *Complexity and approximation of combinatorial algorithms*

Much effort is currently being used in providing theoretical help to construct and to test good algorithms. One especially important point is that clearly stated algorithms and well structured computer programs are much easier to analyse, modify, and to develop further (see e.g. Dahl et al. (1972), Aho et al. (1974) and Dijkstra (1976)); in some cases even the correctness proof may become possible. To choose the appropriate algorithm or data structure we need to compare the different possibilities. For this purpose a measure of complexity is necessary. Two survey papers, Tarjan (1978), and Sahni and Horowitz (1978), extensively discuss the subject, including the definition of the NP-completeness of a problem, techniques for good algorithms, and sample problems for which these techniques are introduced. Both papers contain a long list of references. In addition Tarjan (1978) composes the two measures for complexity: worst-case and average-case behaviour of problem classes. The former is often too pessimistic, the latter has other drawbacks: namely, the lack of good probability measures, the usual supposition of independent data and the surprise of very rare but very bad examples. The paper is concluded by suggested research subjects. Complexity measures of different storage structures are compared in Pippenger and Fischer (1979). Sahni and Horowitz (1978) stress the importance of approximate algorithms and probabilistic algorithms, i.e. those always giving —in a certain sense — very good

solutions and those at a high probability respectively. The paper by Garey and Johnson (1976) consists of an annotated bibliography of approximate algorithms. Further references on combinatorial approximation are: Sahni (1978), Gonzales and Sahni (1978), Nemhouser and Wolsey (1978), Ibarra and Kim (1978). It should be noted however (as mentioned also in the above works), that the asymptotic behaviour of algorithms which are used in complexity measures are not always good indications of the actual behaviour on bounded input. Thus the study of complexity of combinatorial complexity is not a substitute for massive empirical testing but an additional tool for a *priori* analysis.

With regart to the complexity of algorithms, worst case analysis is carried out most often. This may refer to the quality of best solution obtained for heuristic procedures (i.e. the maximal distance from the optimum). Other aims are the number of operations necessary, best performance guarantee (for a limited time) or the maximum amount of computer core memory used. Although computer time and core requirement are convertible to a certain extent, this conversion is not unique and may depend heavily on computer and software facilities. Error bounds are determined for approximate algorithms solving the parallel processor problem (Lam and Sethi (1977)), several different scheduling problems (Garey et al. (1978)), the travelling salesman problem (Rosenkrantz et al. (1977)), and the coin changing problem (Nemhouser and Wolsey (1978)). Most papers discuss extensively the methodology of determining error bounds and they illustrate their quite generally applicable methods on the mentioned examples.

12.1.2 *Probabilistic and asymptotic methods*

Probabilistic approaches are used for different purposes. First of all the mathematical model, i.e. the original problem itself may be probabilistic, e.g. in Halpern (1977) a randomized set covering problem is considered. A chance constrained version (satisfaction with given probability at minimal cost) and a sequential decision variant (repeated choice of variables until satisfaction at minimal expected cost) are solved.

Another approach that of testing and computing the efficiency of different algorithms by random problems. Karp (1977) applies a probabilistic analysis of his partitioning procedure for large travelling salesman problems in the plane. A worst-case error bound is used, the subproblems are solved by a combination of heuristic and exact methods. Stein (1978) solves the single-bus problem of carrying all people between individual origins and destinations. The length of an optimal tour is calculated with high probability for a large number of passengers. The expected performance of an algorithm for the bottleneck assignment problem is given in Garfinkel and Nemhouser (1978).

Dannenbring (1977) outlines a method for estimating the optimum of large combinatorial methods. His asymptotic results are based on statistical sampling and they are illustrated on flow shop sequencing problems.

12.1.3 *Integer polyhedron*

A large number of papers are devoted to the study of the convex hull of integer points of a given polyhedron, called integer polyhedron. The polyhedrality of the convex hull is proved by Meyer and Wage (1978) for the equality case. A collection of papers on polyhedral combinatorics (edited by Balinski and Hoffman, (1978)) deals with questions like characterization of adjacency on 0–1 polyhedra, good characterization of solutions of discrete problems, properties of blocking and antiblacking polyhedra.

Characterizations of all facets of 0–1 polytopes are studied by Balas (1975), Hammer et al. (1975), and Wolsey (1975). A characterization of valid inequalities is found in the framework of superadditivity in Wolsey (1977), and for special structured problems (packing and covering, travelling salesman, etc.) in Wolsey (1977a). Although the faces of an integer polyhedron are the strongest cuts they may be too expensive to calculate compared with other valid cuts. Balas and Zemel (1977) give a necessary and a sufficient condition using critical cutsets of a graph for an inequality to be a facet of the set packing polytope. Glover (1975) provides the means for generating all valid inequalities, as a generalization of disjunctive cuts, by polyhedral annexation, i.e. by building together several convex polyhedra containing feasible integer points only on the surface.

12.2 Advances in general discrete programming algorithms

12.2.1 *Implicit enumeration*

Different enumeration methods are of basic importance even after creating a number of commercial computer codes which are mainly branch-and-bound extentions of linear programming packages. There are two main reasons. First of all in the mentioned packages linear programming is extensively used and this is time consuming. Secondly, often the special character or the combinatorial nature of the problem cannot be taken into account. Logical constraints are hard to handle in linear programming. In the case of pure integer problems it is possible for example to solve the continuous version of the problem only once and to start an enumeration from the continuous optimum. Patterson's (1978) consists of a study of efficiency of five different enumeration procedures using many well-known test problems. There are however many other approaches of mixing enumeration and other types of methods. Burdet and Johnson (1977) suggest a subadditive approach which is a combination of cutting planes and enumeration. Bowman (1977) improves the enumeration by suggesting new partial orderings, one of which is based on the number of non-zero components, the other utilizes the differences of objective function coefficients. Mevert and Stuhl (1977) report the solution of large, structured integer problems mainly by implicit enumeration (using chained storage of nonzero elements) in a reasonable time. Commercial

LP-based codes failed to solve these problems in the rather long allotted time. The generalized upper bounding technique was implicitly used in the program. Guignard and Spielberg (1977) consistently use logical constraints which are derived during the computation.

12.2.2 *Branch-and-bound algorithms*

Such algorithms are widely used to solve both, pure and mixed integer programming problems. At present, all large commercial integer codes – see Johnson and Powell (1978) – are developed from linear programming packages using branch-and-bound principles. As the codes are very effective for continuous linear problems, the extension remains powerful if the number of integer variables is not too high. For large, well structured discrete problems implicit enumeration and the use of logical constraints – developed during the computation – may mean a large improvement.

Another approach is used for many practical applications. If a large number of complicated constraints determine the domain of feasible solutions, then good bounds of the objective function may be obtained by relaxing the problem, i.e. considering only the most essential restrictions. For most well-known practical integer models there are several branch-and-bound approaches (see Section 12.3). This approach makes possible the use of *a priori* information and the development of man–machine systems. A further advantage is obtained if good heuristics provide acceptable solutions because the calculated bounds give the maximal distance from the optimal solution and thus the economy of calculations can also be ensured.

Relatively few new general results have been achieved in the field of branch-and-bound methods, because usually the special structures and corresponding algorithm specifications result in efficient solutions. The theoretical examinations of efficiency measures, dominance relations and the ε-optimality of branch-and-bound procedures are to be found in Ibaraki (1976, 1977a, 1977b); these sources contain long lists of further references. Breu and Burdet (1974) give a detailed description of their computer experimentation with a large number of branch-and-bound rules for 0–1 pure integer problems. Johnson and Powell (1978) provide a survey of integer programming codes, including large scale commercial codes for mixed integer problems, all of which are based on the branch-and-bound principle. This paper also contains many references to other codes for pure integer problems using cutting planes, enumeration, group methods, Lagrangian relaxation or Benders decomposition. Johnson (1978) suggests some ideas (tightenings, relaxations, integer infeasibility) by which branch-and-bound codes can be made more efficient. Shih (1979) effectively solves the multiconstraint 0–1 knapsack problem by a branch-and-bound algorithm using the bounds obtained from knapsack problems of the individual constraints. The method can be extended to a general case in which the constraint coefficients are not restricted in sign. Useful comments appear in Chapter 6 of Kovalev (1977).

12.2.3 *Dynamic programming*

Dynamic programming is a very fast method for linear and separable nonlinear discrete problems if the number of explicit algebraic constraints is small and the coefficients are nonnegative. The amount of computation grows only linearly with the number of variables. Recent research is aimed at combining the method mainly with branch-and-bound ideas and fast heuristics to extend the efficiency of-dynamic programming to problems having many constraints. Good initial feasible solutions play a very important role in quick fathoming. Marsten and Morin (1978) have developed a hybrid algorithm using the above-mentioned components (seef also Morin and Mastern (1976)). It can be viewed as a procedure in the framework of dynamic programming, the excessive number of states is reduced by branch and bound subprocedures, the bounds are obtained by relaxation and fathoming criteria. An opposite view of their algorithm is to take it as a branch-and-bound procedure with dominance for eliminating unnecessary branches by dynamic programming. Dual linear programming is used to provide bounds for all branches simultaneously. A different approach is that of Aust (1976). He first relaxes the problem with several constraints to several problems of one constraint. An initial solution is constructed step by step through equalizing values of variables of the subproblems. The third phase is to obtain an optimal solution by fathoming the solution tree. Dynamic programming is also very promising if there are many logical constraints, with few or no explicit algebraic ones. For example, Baker and Schrage (1978) suggested a new efficient method for solving optimal sequencing problems with precedence relations. A recent paper (Cooper, L. and Cooper, M. W. (1978)) suggests a new, so-called hyperplane search approach limiting the state space and thus solving problems of 10–15 constraints.

12.2.4 *Cutting planes*

Of late, a large number of papers have been devoted to the theory valid inequalities, i.e. to those excluding no feasible integer points but possibly a large part of the originally given continuous region. Jeroslow (1977, 1978) provides extensive surveys of disjunctive cuts and algebraic approaches respectively. The disjunctive approach has several other synonyms: convexity, intersection, geometric. It has been studied by Balas (1975) for logical conditions. The algebraic approach is called subadditive because of the recent emphasis on this property. Burdet (1977–1 1977–2) studies the question of necessary and sufficient conditions for nonconvex optimization within the framework of convex analysis and he determines valid cuts. His polaroid cuts are extended to integer programming. As far as the efficiency of cutting methods are concerned, they are usually combined with other methods (mainly implicit enumeration and branch-and-bound) to avoid pathological cases or to enjoy the advantages of mixed combinatorial–algebraic approaches. See for example Burdet and Johnson (1977), Lev and Soyster (1978), Soyster et al. (1978). Another promising direction is the stepping-stone parallel cut (cf.

Cheung (1978)) which is a combination and extension of Hillier's rounding heuristic procedure and Glover's cut search. There are several cutting plane algorithms for special structured problems: see, for example, Balas (1977) for the set covering, or Miliotis (1978) for the travelling salesman problem.

12.2.5 *Heuristics and approximations*

As the time needed to solve combinatorial (discrete programming) problems increases very rapidly (often exponentially), with some characteristic data (e.g. the number of discrete variables) of the problem the capability of any exact method is limited. For this reason many heuristic methods have been developed. They are usually combined with or imbedded into other algorithms, particularly in the case of special structured problems (see Section 12.3). A comparison of three well-known heuristic procedures may be found in Zanakis (1978). Both the computer time for solving test problems and the error (distance from the optimum) are considered.

Another approach to avoid the combinatorial explosion is to design approximate algorithms. Sahni (1978) presents polynomial time approximations for several well-known NP-complete problems like sequencing with deadlines, 0–1 knapsack, etc. Some experimental results are included. Ibaraki (1976) proved that the efficiency – measured by the number of decomposed branches – of branch-and-bound solutions does not improve in general by requiring ε-optimality. He provides a subclass of problems for which efficiency increases with the inaccuracy vaule ε. Savage (1976) introduces a measure for the effectiveness of local optimization in determining the global optimum of a combinatorial problem: the probability of obtaining the global optimum. His idea is illustrated on a travelling salesman problem and completed with a list of publications on local search methods. Several heuristic methods are discussed in Chapter 7 of Kovalev (1977).

12.3 Development of algorithms for specially structured discrete problems

There is a great variety of problem classes for which algorithms, substantially more efficient than the general ones, exist. The use of quick tests, well fitted and easy to solve relaxations, special bounding procedures, logical implications, etc. may become possible. These factors are very important because of the usually exponential type of growth of computing time as a function of relevant input size, which includes the number of constraints, number of variables, density of data arrays, absolute values and ranges of data, etc.

12.3.1 *Networks and graphs*

Networks and algorithmic graph problems can be formulated also as discrete programming problems. Although conversely it is not true, a network or graph theoretic relaxation can be found to many of the quite general discrete problems.

Network flow or graph theoretic algorithms are often very efficient and even if they are not generally applicable, they may give intuitive insight. On the other hand, general discrete programming algorithms may have to be applied for easy network of graph theoretic problems if some additional constraints must also be satisfied.

The review of network flow and graph theoretic algorithms is beyond the scope of the present book. On algorithmic graph theory the book of Christofides (1976) provides a good view. For flow algorithms the original book of Ford and Fulkerson (1962) is suggested. Network flow procedures are successfully used for solving certain special structured integer problems: for example Glover et al. (1978) solve special linear programming problems (maximum two nonzero entries per column) very quickly; Balas and Samuelson (1977) solve the node-covering problem; Berge and Johnson (1977) apply, linear programming techniques for the edge colouring of hypergraphs; Ibaraki et al. (1976) give an algorithm for an assignment problem on networks that aims to minimize the critical path.

12.3.2 *Knapsack problem*

Even after many years there are still advances for the well-known knapsack problem. Its importance is emphasized by many branch-and-bound type algorithms for more general integer programming problems with knapsack relaxations. A comparison of algorithms is found in Fayard and Plateau (1975). Further algorithms for the basic problem are contained in Nauss (1976), Ingargiola (1977) and Laurier (1978). Large, collapsing problems are discussed in Posner and Guigner (1978). Nonlinear knapsack problems are studied and solved by Morin and Marsten (1976), Michaeli and Pollatschek (1977), and Suzuki (1978). Ibaraki et al. (1978) deal with the basic problem with multiple choice constraints added, i.e. exactly one element from each of the given subgroups of variables must be chosen. The multi-knapsack problem (there are several capacitated knapsacks with one common set of objects) is solved by a branch-and-bound technique using surrogate and Lagrangian relaxations in Hung and Tisk (1978), and by an enumeration method in Shih (1979).

12.3.3 *Scheduling*

Different scheduling problems are extensively studied because of their wide practical applicability and they are of theoretical interest mainly because of their complexity. Low order polynomial algorithms are developed for one, two or three machine sequencing problems with n jobs, in the case of precedence constraints (see Garey and Johnson (1977) Brucker et al. (1977), Adolphson (1977), Nepomiastchy (1978), Kurisu (1977), Baker and Schrage (1978)). The last paper applies dynamic programming. Polynomial approximation algorithms are given by Ibarra and Kim (1978). The performance guarantees (guaranteed objective function value/optimum in worst case) of several approximate algorithms are determined in Garey et al. (1978) for the basic m machine n job sequencing, with

some extensions. Panwalker and Iskander (1977) give a survey of more than a hundred scheduling (dispatching) rules and other heuristic rules. Carlier (1978) solves the general $m \times n$ machine sequencing problem for the case of disjunctive constraints, i.e. when the processing time intervals for some pairs of jobs must be disjunct. Several test examples up to 8 machines are described and solved. The paper of Miyazaki et al. (1978) contains a procedure for the mean flow time scheduling problem. Muckstadt and Koenig (1977) apply Lagrangian relaxation to the scheduling of a power generation system. Branch-and-bound approach is used with different new bounding schemes in Lageweg et al. (1978), and in Bestwick and Hartings (1976).

12.3.4 *Travelling salesman and other routing problems*

For several reasons the travelling salesman problem is one of the most celebrated in discrete programming. The number of feasible solutions obviously grows exponentially with the number of cities. Even the basic problem is challenging, yet both in its original form and in close variations the models describe many real life situations quite well. The entire arsenal of discrete programming methods has already been tried with variable and surprising outcomes.

Implicit enumeration is applied successfully with assignment relaxation for large random test problems in Smith and Thomson (1977) and a lexicographic search (based on a non-decreasing order of city distances) with sequential decisions is used in Gupta (1978). The branch-and-bound method is the basis of Bazaraa and Goode (1977) who use a dual approach with subgradient optimization and of Garfinkel and Gilbert (1978) for the bottleneck travelling salesman problem with probabilistic analysis for large uniformly distributed random problems. The non-deterministic cost problem is examined also by Leipälä (1978) who aims at obtaining minimal expected tour length. The superiority of cutting methods is claimed by Miliotis (1978) over a linear programming based branch-and-bound method Miliotis (1976).

Several different relaxations serve as objective function bounds and/or search directions for better feasible solutions. Minmum spanning 1-trees are used in Smih and Thompson (1977) with LIFO (last-in–first-out) to decrease memory requirement in branch-and-bound search. 1-arborescence is the basis for the algorithm of Garfinkel and Gilbert (1978). Most papers include heuristic rules or procedures. Rosenkrantz et al. (1977) discuss several of these: intersection methods, nearest neighbour methods, k-optimality (k-taple changes) and others together with a complexity analysis.

The applicability of the travelling salesman model is largely increased by introducing additional constraints like the existence of clusters (subsets of cities) which must be visited contiguously. Furthermore, specific pairs of cities must be reached in a given order. These restrictions are treated in Lokin (1979). Some others for multiple salesman (8 hour working days, capacities of transport vehicle, etc.) are discussed in Dinkel et al. (1976).

In the general routing problems similar questions should be answered as in the travelling salesman problem but the single salesman (or vehicle) is now replaced by several of them. A survey on recent development for vehicle routing is given by Golden (1978). Usually the capacities of vehicles are restricted as in Holmes and Parker (1976) and Christofides (1976). The latter gives an exact and several heuristic algorithms based on enumeration, set partitioning and the travelling salesman, respectively. The generalized savings criterion is applied in Foster and Ryan (1976), and in Mole and Jameson (1976). A stochastic variant is examined by Stein (1978), supposing transport from random locations to random locations. An asymptotic analysis is also given for the expected length of a minimal tour. Several polynomial time approximate algorithms are presented for NP-complete routing problems in Frederickson et al. (1978) with worst case bounds on the distance from the optimum.

12.3.5 *Set covering, partitioning and packing*

An important special case of 0–1 linear integer problems is when all the non-zero constraint coefficients and all right hand side values are 1. All the constraints have $\geqq$, $=$, $\leqq$ types of relations. Besides their direct applicability these problems can be considered as relaxations of more general discrete problems. They can be effectively used to represent complicated logical relations. They can be solved by any of the methods of discrete programming, but substantially more efficient ones can also be developed using their special structures. An extensive survey of set partitioning problems including the examination of the related polytope, algorithms and applications is provided by Balas and Padberg (1976). Valid inequalities for this problem are discussed further by Balas (1977). Etcheberry (1977) solves the set covering problem by implicit enumeration using Lagrangian multipliers and subgradient optimization. He claims that this is much more efficient than using the simplex method for obtaining bounds. Houck and Vemuganti (1977) apply an enumeration algorithm for the group theoretic structure of the vertex packing problem after obtaining good initial solutions with the help of a related bipartite graph.

12.3.6 *Quadratic assignment problem*

This kind of problem provides a useful model for many facility location or other placement problems when discrete location costs and costs of interaction (e.g. transportation) occur between pairs of facilities. The latters are quadratic functions of the location variables. Hanan and Kurtzberg (1972) provide a detailed survey of the earlier results. Numerical investigations, i.e. comparison on many approximate algorithms on test examples, can be found in Parker (1976), and in Burkhard and Stratman (1978). The simple plant location model is applied in Kauffman et al. (1977). Bilde and Krarup (1977) determine sharp lower bounds and an efficient algorithm which can also be applied for the set covering problem. Guignard and

Spielberg (1977) exploit the special structure of the problem and establish a decomposition procedure with the modified simplex method. A parallel production problem with changeover costs is transformed into a quadratic assignment problem in Geoffrion and Graves (1976). Pre-partitioning is applied in Patel et al. (1976). Akine and Khumawala (1977) give a branch-and-bound method for the capacitated warehouse problem, which is improved by Nauss (1978) using Lagrangian relaxation. Krarup and Pruzan (1978) suggest approximate methods for computer aided layout design with criticism and a suggestion of an interactive system.

12.3.7 *Further typical applications*

The single model assembly line balancing problem is solved by a heuristic network method in Pinto et al. (1978), by heuristic dynamic programming in Davis and Simmons (1977) and by a branch-and-bound method using dominance rules and a reliable branching heuristic in van Assche and Herroelen (1979). Mixed model problems, i.e. the case of several different products on the assembly line, are studied by Dar-El (1978). A two-dimensional cutting problem is solved by Christofides and Whitlock (1977) and a heuristic procedure is suggested for a nonlinear cutting stock problem by Coverdale and Wharton (1976). An investment planning problem is treated by a mixed integer programming model in van Hulst and van Lieshout (1978). Data set allocation problems and suggested approximate algorithms are given in Wong (1976). Dunstan (1977) provides an approximate solution for the general resource allocation problem with worst case bound on the optimum and optimality proven for a special case. A polynomial time algorithm is given for the resource allocation problem with convex objective function in Katoh et al. (1979). A combinatorial optimization formulation and implicit enumeration type of algorithm (and an approximate one for large problems) is given for road network planning with practical application in Anzai (1977). A fixed charge transportation problem is handled by a branch-and-bound method in Kennington and Unger (1976). A loading problem – known as bin-packing – is solved using a highly efficient method by Hung and Brown (1978).

12.4 Extensions, experimentation

12.4.1 *Sensitivity and parametric analysis, duality*

Sensitivity and parametric analysis of integer programming problems have special importance in addition to modelling problems as in linear programming. They can be used to overcome the difficulty of non-continuity of optimum as a function of input data and they may provide a resource evaluation.

Geoffrion and Nauss (1977) give an account of the purposes and tools for parametric and postoptimality analysis in integer programming. In their approach a mixed integer problem is embedded into a family of problems (with one param-

eter to change either the objective function or the right-hand side). The paper aims at studying the stability of the optimal solution, the change of optimum as a function of the parameter, and the examination and modification of branch-and-bound codes in order to solve parametric and postoptimality problems effectively. Shapiro (1977) also uses the duality of linear programming to solve parametric problems with changing objective function coefficients or right-hand sides or matrix coefficients. He formulates an extension of complementary slackness for integer problems. Bell and Shapiro (1977) suggest a finite series of dual problems with decreasing duality gaps at the end of which the optimal solution of the integer problem is obtained. In other words the integer polyhedron of the original problem is approximated in the neighbourhood of an optimal solution by Gomory cuts. Holm and Klein (1978) apply shifted Gomory cuts for parametric right hand side changes. Piper and Zoltners (1976) modify the usual branch-and-bound method by loosening the fathoming condition in order to obtain near-optimal solutions, containing the optimal solutions of a wide variety of perturbed problems. Marsten and Morin (1977) solve the parametric right-hand side integer problems by introducing incumbents for intervals of the parameter in the branch-and-bound procedure.

A somewhat similar approach is used in Glover (1978) for a different purpose: the parametric analysis results in a cost evaluation of constraints (both the integrality and the inequality constraints can be examined). This is likely to be very helpful both for modelling and in the course of problem solving.

Fischer et al. (1975) discuss the use of duality in solving integer programming problems: obtaining strong bounds for the objective function, finding feasible solutions, guiding the search in enumeration schemes. Lovász (1977) is mainly devoted to primal–dual pairs of integer problems with no duality gap.

12.4.2 *Nonlinear integer problems*

Even the linear discrete programming problems are usually difficult to solve, in contrast to the continuous case. There is no big gap between linear and nonlinear cases. The additional difficulty usually comes from the continuous relaxation.

Different ways of decomposing discrete convex problems into a series of combinatorial and continuous convex problems are studied by Geoffrion (1977). Each of his four suggestions is illustrated by a real life model and its solution. A theoretical basis for convex integer problems is to be found in Belousov (1977). Convex separable objective functions are considered by Michaeli and Pollatschek (1977) with a single linear equation. Separable nonlinear multi-dimensional knapsack problems are solved by Morin and Marsten (1976). In their dynamic programming approach dominance of solutions is used to reduce the problem to a one-dimensional one. Numerical experimentation is given. Abadie et al. (1976) compare two versions of branch-and-bound procedures on seven nonlinear test examples.

Phuong (1976) describes an algorithm with a modified Beale method as a relaxation for problems with quadratic objective function and linear constraints; test problems are included. Glover (1975) suggests an improved linear formulation for nonlinear, mainly quadratic integer, problems. Coverdale and Wharton (1976) give a heuristic procedure for nonlinear cutting stock problems.

12.4.3 *Miscellaneous*

An important point in developing efficient algorithms is testing. Besides the numerous test problems in the literature, e.g. Garfinkel and Nemhouser (1972), there are test problem generating procedures, see Lin and Rardin (1977), and Fleischer and Meyer (1978). The latter paper also provide sufficient condition for optimality. There are strong efforts to increase efficiency by interactive methods and it is done on different level of generality.

Guignard and Spielberg (1978) have designed an experimental interactive integer programming system. Thomas et al. (1978) have solved a chemical blending problem via on-line integer programming. Daniel (1978) shows the way of reducing computational effort on a single but large discrete problem.

Lagrangian relaxation is discussed extensively in Geoffrion (1974) and a good method of determining optimal multipliers is also given. Gavish (1978) aims at reducing the computing effort for the repeated determining of these multipliers.

Piehler (1977) uses congruences in solving integer programming problems. An upper bound is given for the number is integer solutions of an equation in Lambe (1977). Bachem (1977) decomposes the integer problem on a cone. Chvátal and Hammer (1977) present a subclass of problems when an equivalent problem can be obtained by aggregating inequalities without size increase.

McDaniel and Devine (1977) give a modified Benders decomposition procedure. Gauthier and Ribière (1977) discuss heuristic branch-and-bound methods using the concept of pseudo-cost.

Abadie, J., Dayan, H. and Akoka J. (1976) Quelques Experiences Numériques sur la Programmation Non-Linéaire en Nombre Entiers. *RAIRO Rech. Opérat. 10*, 65–70.

Adolphson, D. L. (1977) Single Machine Job Sequencing with Precedence Constraints. *SIAM J. Comput. 6*, 40–54.

† Aho, A. V., Hopcroft, J. E. and Ullman J. D. (1974) *The Design and Analysis of Computer Algoritms*, Addison-Wesley.

Aiello, A., Burattini, E. and Massarotti, A. (1977) Reducibility as a Tool to Extend the Power of Approximation Algorithm. The Minimization of Boolean Expressions. *RAIRO Inf. théor., 11*, 75–82.

Akine, U. and Khumawala B. M. (1977) An Efficient Branch-and-Bound Algorithm for the Capacitated Warehouse Location Problem, *Management Science, 23*, 585–594.

Andrews, G. E. (1974) Asymptotic Methods in Enumeration, *SIAM Review 16*, 485–515.

Anzai, Y. (1977) Urban Road Network Modelling Problem—Formulation and Algorithms, *J. Op. Res. Soc. Japan 20*, 203–230.

†Ashour, S. (1972) Sequencing Theory, *Lecture Notes in Economics and Mathematical Systems, 69*.

van Assche, F. and Herroelen, W. S. (1979) An Optimal Procedure for the Single-Model Deterministic Assembly Line Balancing Problem, *EJOR, 3*. 142–149.

d'Atri, G. (1978) Improved Lower Bounds to the Travelling Salesman Problem, *RAIRO Rech, Opérat, 12*, 369–382.

Aust, R. J. (1976) A Dynamic Programming Branch and Bound Algorithm for Pure Integer Programming. *Comput. & Ops. Res. 3*, 27–38.

Bachem, A. (1977) Reduction and Decomposition of Integer Programs over Cones. *Annals of Discrete Mathematics 1*, 1–11.

†Baker, K. R. (1974) *Introduction to Sequencing and Scheduling*. John Wiley, New York.

Baker, K. R. and Schrage, L. E. (1978) Finding an Optimal Sequence by Dynamic Programming: An Extension to Precedence-Related Tasks, *Operations Research, 26*, 111–120.

Balas, E. (1975a) Facets of the Knapsack Polytope. *Math. Progr. 8*, 146–164.

Balas, E. (1975b) Disjunctive Programming: Cutting Planes from Logical Conditions, in O. Mangasarian, R. R. Mayer and S. Robinson (eds.) *Nonlinear Programming, 2*, Academic Press, 279–311.

Balas, E. (1977) Some Valid Inequalities for the Set Partitioning Problem. *Annals of Discrete Mathematics 1*, 13–47.

Balas, E. and Padberg, M. W. (1975) On the Set Covering Problem II. An Algorithm for Set Partitioning, *Operations Research, 23*, 74–90.

Balas, E. and Padberg, M. W. (1976) Set Partitioning: A Survey, *SIAM Review, 18*, 710–760.

Balas, E. and Samuelsson, H. (1977) A Node Covering Algorithm. *Naval Research Logistics Quarterly, 24*, 213–233.

Balas, E. and Zemel E. (1977) Critical Cutsets of Graphs and Canonical Facets of Set-Packing Polytopes. *Math. Op Res. 2*, 15–19.

† Denotes book or collection of papers.

Balinski, M. L. ed. (1974) Approaches to Integer Programming. *Math. Programming, 2,* 1–197.
Balinski, M. L. Hoffman, A. J. eds (1978) Polyhedral Combinatorics, *Math. Progr, 8,* 1–234.
Balinski, M. L. and Russakoff, A. (1974) On the Assignment Polytope. *SIAM Review, 16,* 516–525.
Bartholdi III, J. J. and Rattiff, H. D. (1978) Unnetworks, with Application to Idle Time Scheduling, *Management Sci., 24,* 850.
Bazaraa S. and Goode, J. J. (1977) The Traveling Salesman Problem: A Duality Approach, *Math. Progr. 13,* 221–237.
Bell, D. F. and Shapiro, J. F. (1977) A convergent Duality Theory for Integer Programming, *Operations Research, 25,* 419–434.
Belousov, E. G. (1977) Vvegenije v Vuepukluej Analiz i Celochislennoje Programirovanije (Introduction to Convex Analysis and Integer Programming) (Russian) Izdatelstvo Moskoskovo Universiteta.
Bestwick, P. F. and Hartings, N. A. J. (1976) A New Bound for Machine Scheduling, *Operations Research Quarterly, 27,* 479–488.
Berge, C. and Johnson, E. L. (1977), Coloring the Edges of a Hypergraph and Linear Programming Techniques. *Annals of Discrete Mathematics, 1,* 65–78.
Bilde, O. and Krarup, J. (1977), Sharp Lower Bounds and Efficient Algorithms for the Simple Plant Location Problem. *Annals of Discrete Mathematics, 1,* 79–98.
Blair, C. E. and Jeroslow, R. G. (1977) The Value Function of a Mixed Integer Program I. *Discrete Mathematics, 19,* 121–138.
Blair, C. E. and Jeroslow, R. G. (1979) The Value Function of a Mixed Integer Program II. *Discrete Mathematics, 25,* 7–20.
†Boffey, T. B., ed. (1977) *Proceedings of CP 77,* Combinatorial Programming Conference, Liverpool.
Bonney, M. C. and Gundry, S. W. (1976) Solutions to the Constained Flow-Shop Sequencing Problem. *Operational Research Quarterly, 27,* 869–884.
Bowman, V. J. (1977) Partial Orderings in Implicit Enumeration, *Annals of Discrete Mathematics. 1,* 99–116.
Breu, R. and Burdet, C.-A. (1974) Branch-and-Bound Experiments in Zero–One Programming, *Math. Progr. 2,* 1–50.
Brucker, P., Garey, M. R. and Johnson, D. S. (1977) Scheduling Equal-Length Tasks under Tree-Like Precedence Constraints to Minimize Maximum Lateness, *Math. Op. Res. 2,* 275–284.
Burder, C. A. and Johnson, E. L. (1977) A Subadditive Approach to Solve Linear Integer Programs, *Annals of Discrete Mathematics, 1,* 117–143.
Burdet, C. (1977–1) Elements of a Theory in Non-Convex Programming, *Naval Research Logistics Qarterly* 47–66.
Burdet, C. (1977–2) Convex and Polaroid Extensions, *Naval Research Logistics Quarterly, 24,* 67–82.
Burkhard, R. E. and Stratman, K. (1978) Numerical Investigation on Quadratic Assigment Problems, *Naval Research Logistics Quarterly, 25,* 129–148.

Carlier, J. (1978) Ordonnancement a contraintes disjonctive, *RAIRO Rech. Opérat., 12,* 331–350.
Chen, D.-S. and Zionts, S. (1976) Comparison of Some Algorithms Solving the Group Theoretic Integer Programming Problem. *Operations Research, 24,* 1120–1128.
Cheung, T. (1978) A Stepping – Stone Parallel Cut Method for Integer Programming. In *Computers and Mathematical Programming,* White ed. 31–37.
†Christofides, N. (1975) *Graph Theory: An Algorithmic Approach.* Academic Press, New York.
Christofides, N. (1976) The Vehicle Routing Problem. *RAIRO Rech. Opérat., 10,* 55–70.
Christofides, N. and Whitlock, Ch. (1977) An Algorithm for Two-Dimensional Cutting Problems. *Operations Research, 25,* 30–44.

Chvátal, V. and Hammer, P. L. (1977) Aggregation of Inequalities in Integer Programming. *Annals of Discrete Mathematics, 1*, 145–162.
Coffman, Jr., E. G. and Sethi, R. (1976) A Generalized Bound on LPT Sequencing, *RAIRO Inf., 10*, 17–26.
Cooper, L. and Cooper, M. W. (1978) All Integer Linear Programming — A New Approach Via Dynamic Programming, *Naval Research Logistics Quarterly, 25*, 415–430.
Coverdale, I. L. and Wharton, F. (1976) An Improved Heuristic Procedure for a Nonlinear Cutting Stock Problem, *Management Sci. 23*, 78–86.

† Dahl, O. J. Dijkstra, E. W. and Hoare, C. A. R. (1972) *Structured Programming*. Academic Press, New York.
Daniel, R. C. (1978) Reducing Computational Effort in Solving a Hard Integer Program, *SIGMAP Bulletin 24*, 33–36.
McDaniel, D. and Devine, M. (1977) A Modified Bender's Partitioning Algorithm for Mixed Integer Programming. *Management Sci., 24*, 312–319.
Dannenbring, D. G. (1977) Procedures for Estimating Optimal Solution Values for Large Combinatorial Problems. *Management Sci., 23*, 1273–1283.
Dar-El, E. M. (1978) Mixed-Model Assembly Line Sequencing Problems, *OMEGA, 6*, 313–324.
Davis, K. R. and Simmons, L. F. (1977) Improving Assembly Line Efficiency: A Dynamic Programming Heuristic Approach, *Comput & Op. Res., 4*, 75–88.
†Dijkstra, E. W. (1976) *A Discipline of Programming*. Prentice-Hall, Englewood Cliffs, N. J.
Dinkel, J. J., Kleindorfer, G. B., Kochenberger, and Wong, S.-N. (1976) Environmental Inspection Routes and the Constrained Traveling Salesman Problem, *Comput & Op. Res., 3*, 269–282.
Dunston, F. D. J. (1977) An Algorithm for Solving a Resource Allocation Problem. *Operational Research Quarterly, 28*, 839–857.

†Elmaghraby, S. E., ed. (1973) Symposium on the Theory of Scheduling and its Application. *Lecture Notes in Economics and Mathematical Systems. 86*, Springer, New York.
Etcheberry, J. (1977) The Set-Covering Problem: A New Implicit Enumeration Algorithm, *Operations Research, 25*, 760–772.
Even, S. and Tarjan, R. E. (1976) A Combinatorial Problem Which is Complete in Polynomial Space, *J. Assoc. Comp. Mach., 23*, 710–719.

Fayard, D. and Plateau, G. (1975) Resolution of the 0–1 Knapsack Problem, Comparison of Methods. *Math. Progr. 8*, 272–307.
Fischer, M. L., Northup, W. D. and Shapiro, J. F. (1975) Using Duality to Solve Discrete Optimization Problems: Theory and Computational Experience. *Math. Progr. 3*, 56–94.
Fleischer, J. M. and Meyer, R. R. (1978) New Sufficient Optimality Conditions for Integer Programming and their Application. *Comm. Assoc. Comp. Mach. 21*, 411–418.
†Ford, L. R. and Fulkerson, D. R. (1962) *Flows in Networks*. Princeton, Princeton University Press, New Jersey.
Foster, B. A. and Ryan, D. M. (1976) An Integer Programming Approach to the Vehicle Scheduling Problem. *Operational Research Quarterly, 27*, 367–384.
Fox, B. L., Lenstra, J. K. Rinnoy Kan, A. H. G. and Scharge, L. E. (1970) Branching from the Largest Upper Bound: Folklore and Facts, *EJOR, 2*, 191–194.
Frederickson, G. N., Hecht, M. S. and Kim, Ch. E. (1978) Approximation Algorithms for Some Routing Problems. *SIAM J. Comput. 7*, 178–193.

Garey, M. R. and Johnson, D. S. (1976) Approximation Algorithms for Combinatorial Problems: An Annotated Bibliography, *Algorithms and Complexity: Recent Results and New Directions*, Traub, ed. 41–52.
Garey, M. R. and Johnson, D. S. (1977) Two Processor Scheduling with Start-Times and Deadlines. *SIAM J. Comput., 6*, 416–426.

Garey, M. R. ,Johnson, D. S. and Sethi, R. (1976) The Complexity of Flowshop and Jobshop Scheduling. *Math. Op. Res.*, *1*, 117–129.

Garey, M. R., Graham, R. L. and Johnson, D. S. (1978) Performance Guarantees for Scheduling Algorithms. *Operations Research*, *26*, 3–31.

†Garfinkel, R. S. and Nemhouser, G. L. (1972) *Integer Programming*, John Wiley.

Garfinkel, R. S. and Gilbert, K. C. (1978) The Bottleneck Traveling Salesman Problem: Algorithms and Probabilistic Analysis. *J. Assoc. Comp. Mach. 25*, 435.

Gauthier, J. M. and Ribière, G. (1977) Experiments in Mixed-Integer Linear Programming Using Pseudo-Costs. *Math. Progr. 12*, 26–47.

Gavish, B. (1978) On Obtaining the "Best" Multipliers for a Lagrangian Relaxation for Integer Programming, *Comput & Op. Res.*, *5*, 55–72.

Geoffrion, A. M., (1974) Lagrangian Relaxation for Integer Programming, *Math. Progr. 2*, 82–114.

Geoffrion, A. M. (1976) A Guided Tour of Recent Practical Advances in Integer Linear Programming. *OMEGA*, *4*, 49–58.

Geoffrion, A. M. (1977) How can Specialized Discrete and Convex Optimization Methods be Married. *Annals of Discrete Mathematics*, *1*, 205–220.

Geoffrion, A. M. and Graves, G. W. (1976) Scheduling Parallel Production Lines with Changeover Costs: Practical Application of a Quadratic Assignment /LP Approach, *Operations Research*, *24*, 595–611.

Geoffrion, A. M. and Nauss, R. (1977) Parametric and Postoptimality Analysis in Integer Programming. *Management Sci.*, *23*, 453–466.

Glover, F. (1975) Polyhedral Annexation in Mixed Integer and Combinatorial Programming. *Math. Progr. 9*, 161–168.

Glover, F. (1978) Parametric Branch and Bound. *OMEGA*, *6*, 145–152.

Glover, F. (1975) Improved Linear Integer Programming Formulations of Nonlinear Integer Problems. *Management Sci.*, *22*, 455–460.

Glover, F., Hultz, D., Klingman, D. and Stitz, J. (1978) Generalized Networks: A Fundamental Computer-Based Planning Tool. *Management Sci.*, *24*, 1209–1220.

Glover, F. and Klingman, D. (1978) Modelling And Solving Network Problems. *Design and Implementation of Optimization Software. Geenberg, ed.* 185–224.

Golden, B. L. (1978) Recent Developments in Vehicle Routing. *Computers and Mathematical Programming, White, ed.* 233–240.

Gonzalez, T. and Sahni, S. (1978) Flowshop and Jobshop Schedules: Complexity and Approximation. *Operations Research, 26*, 36–52.

†Greenberg, H. J. ed. (1978) *Design and Implementation of Optimization Software*. Sijthoff & Noordhoff, Alpen aan den Rijn, The Netherlands.

Greenberg, H. J. (1978) A Tutorial on Matricial Packing. *Design and Implementation of Optimiration Software. Greenberg, ed.* 109–142.

Guignard, M. and Spielberg, K. (1978) An Experimental Interactive System for Integer Programming. *Computers and Mathematical Programming, White, ed.* 328–337.

Guignard, M. and Spielberg, K. (1977) Algorithms for Exploiting the Structure of the Simple Plant Location Problem. *Annals of Discrete Mathematics*, *1*, 247–272.

Guignard, M. and Spielberg, K. (1977) Reduction Methods for State Enumeration Integer Programming. *Annals of Discrete Mathematics*, *1*, 273–285.

Gupta, J. N. D. (1976) A Heuristic Algorithm for the Flowshop Scheduling Problems. *RAIRO Rech. opérat., 10*, 63–74.

Gupta, J. N. D. (1978) A Search Algorithm for the Traveling Salesman Problem. *Comput & Op. Res.*, *5*, 243–250.

Halpern, J. (1977) The Sequential Covering Problem under Uncertainty. *INFOR*, *15*, 76–93.

†Hammer, P. L., Johnson, E. L., Korte, B. H. and Nemhouser, G. L. (eds) (1977) Studies in Integer Programming. *Annals of Discrete Mathematics*, *1*, North-Holland.

Hammer, P. L., Johnson, E. L. and Peled, U. N. (1975) Facets of Regular 0–1 Polytopes. *Math. Progr. 8*, 179–206.

†Hammer, P. L. and Zoutendijk, G. (eds) (1974) *Mathematical Programming in Theory and Practice*. North-Holland.

Hanan, M. and Kurtzberg, J. M. (1972) A Review of the Placement and Quadratic Assignment Problem. *SIAM Review, 14*, 324–342.

Holm, S. and Klein, D. (1978) Discrete Right Hand Side Parametrization for Linear Integer Programs. *EJOR, 2*, 50–53.

Holmes, R. A. and Parker, R. G. (1976) A Vehicle Scheduling Procedure Based Upon Savings and Solution Perturbation Schemes. *Operations Research Quarterly 27*, 83–92.

Houck, D. J. and Vemuganti, R. R. (1977) An Algorithm for the Vertex Packing Problem. *Operations Research, 25*, 773–787.

† Hu, T. C. and Robinson, S. M., eds (1973) *Mathematical Programming*, Academic Press.

van Hulst, W. G. and van Lieshout, J. Th. (1978) Investment/Financial Planning with Endogeneus Lifetimes: A Heuristic Approach to Mixed Integer Programming. *RAIRO Rech, Opérat., 12*, 85–102.

Hung, M..S. and Brown, J. R. (1978) An Algorithm for a Class of Loading Problems. *Naval Research Logistics Quarterly, 25*, 289–298.

Hung, M. S. and Tisk, J. C. (1978) An Algorithm for 0–1 Multiple-Knapsack Problems. *Naval Research Logistics Quarterly, 25*, 571–579.

Ibaraki, T. (1976a) Integer Programming Formulation of Combinatorial Optimization Problems. *Discrete Mathematics, 16*, 39–52.

Ibaraki, T. (1976b) Computational Efficiency of Approximate Branch-and-Bound Algorithms. *Math. Op. Res. 1*, 287–298.

Ibaraki, T. (1977a) The Power of Dominance Relations in Branch-and-Bound Algorithms, *J. Assoc. Comp. Mach. 24*, 264–279.

Ibaraki, T. (1977b) On the Computational Efficiency of Branch-and-Bound Algorithms. *J. Opns. Res. Soc. Japan, 29*, 16–35.

Ibaraki, T., Hasegawa, T., Teranaka, K. and Iwase, J. (1978) The Multiple-Choice Knapsack Problem. *J. Opns. Res. Soc. Japan, 21*, 59–93.

Ibaraki, T., Ishii, H. and Mine, H. (1976) An assignment Problem on a Network. *J. J. Opns. Res. Soc. Japan, 19*, 70–90.

Ibarra, O. H. and Kim, Ch. E. (1978) Approximation Algorithms for Certain Scheduling Problems. *Math. Op. Res., 3*, 197–204.

Ingargiola, G. P. (1977) A General Algorithm for the One-Dimensional Knapsack Problems. *Operations Research, 25*, 725–759.

Jeroslow, R. G. (1977) Cutting Plane Theory: Disjunctive Methods. *Annals of Discrete Mathematics, 1*, 293–330.

Jeroslow, R. G. (1978) Some Basis Theorems for Integral Monoids. *Math. Op. Res., 3*, 145–154.

Jeroslow, R. G. (1978) Cutting-Plane Theory: Algebraic Methods. *Discrete Math., 23*, 121–150.

Johnson, E. L. (1978) Some Considerations in Using Branch-and-Bound Codes. *Design and Implementation of Optimization Software. Greenberg, ed.* 241–247.

Jonhson, E. L. and Powell, S. (1978) Integer Programming codes. *Design and Implementation of Optimization Software. Greenberg, ed.* 225–240.

Juel, H. and Love, R. F. (1976) An Efficient Computational Procedure for Solving the Multi Facility Rectilinear Facility Location Problem. *Operational Research Quarterly, 27*, 697–704.

Karp, R. M. (1977) Probabilistic Analysis of Partitioning Algorithms for the Traveling Salesman Problem in the Plane. *Math. Op. Res., 2*, 209–224.

Karp, R. M. (1972) Reducibility of Combinatorial Problems. In *Complexity of Computer Computations*. R. E. Miller and J. W. Thatcher (eds) Plenum Press, New York, 85–103.
†Kasting, C. (ed.) (1976) *Integer Programming and Related Areas: A Classified Bibliography*. Springer, New York.
Katoh, N., Ibaraki, T. and Mine, H. (1979) A Polynomial Time Algorithm for the Resource Allocation Problem with a Convex Objective Function. *J. Opns. Res. Soc. Japan, 39*, 449–456.
†Kaufman, A. and Henry-Labordère, A. (1977) *Integer and Mixed Programming: Theory and Applications*. Academic Press, New York.
Kauffman, L., Vanden Eede, M. and Hansen, P. (1977) A Plant and Warehouse Location Problem. *Operational Research Quarterly, 28*, 547–554.
Kennington, J. and Unger, E. (1976) A New Branch-and-Bound Algorithm for the Fixed-Charge Transportation Problem. *Management Sci., 22*, 1116–1126.
†Korbut, A. A., and Finkelstein, Yu. (1969) Diskretnoje programirovanie (Discrete Programming) (Russian). Nauka, Moskva.
Kovaljev, M. M. (1977) Diskretnaja Optimizacija (Discrete Optimization) (Russian) Minsk, Izdatelstvo BGU in. V. I. Lenina.
Krarup, J. and Pruzan, P. M., (1978) Computer Aided Layout Design, *Math, Progr. 9*, 75–94.
Kuba, H. and Okina, N. (1976) A New Practical Solution for the Large-Scale Traveling Salesman Problem. *J. Opns. Res. Soc. Japan, 19*, 272–285.
Kurisu, T. (1977) Three-Machine Scheduling Problems With Precedence Constraints. *J. Opns. Res. Soc. Japan, 20*, 231–242.

Lageweg, B. J., Lenstra, J. K. and Rinnooy Kan, A. H. G. (1978), A General Bounding Scheme for the Permutation Flow-Shop Problem. *Operations Research, 26*, 53–67.
Lam, Sh. and Sethi, R. (1977) Worst Case Analysis of Two Scheduling Problems. *SIAM J. COMPUT 6*, 518–536.
Lambe, T. A. (1977) Upper Bounds on the Number of Nonnegative Integer Solutions to a Linear Equation. *SIAM J. Appl. Math., 32*, 215–219.
Laurier, M., (1978) An Algorithm for the 0/1 Knapsack Problem. *Math. Progr. 14*, 1–10.
Lawler, E. L. (1976) Sequencing to Minimize the Weighted Number of Tardy Jobs. *RAIRO Inf. 10*, 27–35.
Leipälä, T. (1978) On the Solution of Stochastic Traveling Salesman Problems. *EJOR 2*, 291–297.
Lenstra, J. K. and Rinnooy Kan, A. H. G. (1978) On the Expected Performance of Branch-and-Bound Algorithms, *Operations Research, 26*, 347–358.
Lenstra, J. K. and Rinnooy Kan, A. H. G. (1978) Complexity of Scheduling under Precedence Constraints. *Operations Research, 26*, 22–35.
Lev, B. and Soyster, A. L. (1978) Integer Programming with Bounded Variables via Canonical Separation. *J. Opl. Res. Soc., Japan 29*, 477–488.
Lin, B. W. Y. and Rardin, R. L. (1977) Development of a Parametric Generating Procedure for Integer Programming Test Problems. *J. Assoc. Comp. Mach. 24*, 465–472.
Lokin, F. C. J. (1979) Procedures for Traveling Salesman Problems with Additional Constraints. *EJOR 3*, 135–141.
Lovasz, L. (1977) Certain Duality Principles in Integer Programming. *Annals of Discrete Mathematics, 1*, 363–374.

Marsten, R. E. (1977) Parametric Integer Programming: The Right Hand Side Case. *Annals of Discrete Mathematics, 1*, 375–390.
Marsten, R. E. and Morin, T. L. (1978) A Hybrid Approach to Discrete Mathematical Programming. *Math. Progr. 14*, 21–40.
Mevert, P. and Stuhl, U. (1977) Implicit Enumeration with Generalized Upper Bounds. *Annals of Discrete Mathematics 1*, 393–402.

Meyer, R. R. and Smith, M. L., (1978) Algorithms for a Class of "Convex" Nonlinear Integer Programs. *Computers and Mathematical Programming, White, ed.* 210–215.

Meyer, R. R. and Wage, M. L. (1978) On the Polyhedrality of the Convex Hull of the Feasible Set of an Integer Program. *SIAM J. Control and Opt., 16,* 682–687.

Michaeli, I. and Pollatschek, M. A. (1977) On Some Nonlinear Knapsack Problem. *Annals of Discrete Mathematics, 1,* 403–414

Miliotis, P. (1976) Integer Programming Approaches to the Traveling Salesman Problem. *Math. Progr. 10,* 367–378.

Miliotis, P., (1978) Using Cutting Planes to Solve the Symmetric Traveling Salesman Problem. *Math. Progr. 15,* 177–180.

Minoux, M. (1977) Une application de la notion de dualité en programmation en nombres entiers: sélection et affectation optimale d'une flotte d'avion. *RAIRO Rech. Opérat, 11,* 201–222.

Miyazaki, S., Nishiayama, N. and Hashimoto, F. (1978) An Adjacent Pairwise Approach to the Mean Flow-Time Scheduling Problem. *J. Opns. Res. Soc. Japan, 21,* 287–299.

Mole, R. H. and Jameson, S. R. (1976) A Sequential Route-Building Algorithm Employing a Generalised Savings Criterion. *Operational Research Quarterly, 27,* 503–512.

Morin, T. L. and Marsten, R. E. (1976a) Branch-and-Bound Strategies for Dynamic Programming. *Operations Research, 24,* 611–1627.

Morin, T. L. and Marsten, R. E. (1976b) An Algorithm for Nonlinear Knapsack Problems, *Management Sci., 22,* 1147–1158.

Morton, T. E. and Dharan, B. G. (1978) Algorithms for Single-Machine Sequencing with Precedence Constraints. *Management Sci., 24* 1011–1020.

Muckstadt, J. A. and Koenig, Sh. A. (1977) An Application of Lagrangian Relaxation to Scheduling in Power-Generation Systems. *Operations Research, 25,* 387–403.

†Murthy, K. G. (1976) *Linear and Combinatorial Programming,* Wiley, New York

Nauss, R. M. (1976) An Efficient Algorithm for the 0–1 Knapsack Problem. *Management Sci., 23,* 27–31.

Nauss, R. M. (1978) An Improved Algorithm for the Capacitated Facility Location Problem. *J. Opl. Res. Soc. Japan, 29,* 1195–1202.

Nepomiastchy, P. (1978) Rèsolution d'un problème d'ordonnancement à resources variables. *RAIRO, Rech. opérat., 12,* 249–262.

Nemhouser, G. L. and Wolsey, L. A. (1978) Best Algorithms for Approximating the Maximum of a Submodular Set Functions. *Math. Op. Res., 3,* 177–188.

Panwalkar, S. S. and Iskander, W. (1977) A Survey of Scheduling Rules. *Operations Research, 25,* 45–61.

Parker, C. S. (1976) An Experimental Investigation of Some Heuristic Strategies for Component Placement, *Operational Research Quarterly, 27,* 71–82.

Patel, A. M., De Wald, C. G. and Cotes, L. C. (1976) Pre-partitioning as a Means of Enhancing Pairwise Interchange Solutions to Quadratic Assignment Problems. *Comput & Op. Res. 3,* 49–56.

Patterson, J. H. (1978) Balasian-Based Enumeration Procedures. *Computers and Mathematical Programming, White, ed.* 241–250.

Phuong, T. H. (1976) Solution of Integer Programs with a Quadratic Objective Function. *J. Assoc. Comp. Mach. 23,* 468–474.

Piehler, J. (1977) Ganzzahlige Optimierung mit Kongruenznebenbedingungen. *Math. Operationsforsch. Stat. Ser. Opt. 8,* 147–150.

Piehler, J. (1978) Zur Drehung von Gomory Schnitten. *Math. Operationsforsch. Stat. Ser. Opt., 9,* 3–8.

Pinto, P. A., Donnenbring, D. G. and Khumawala, B. M. (1978) A Heuristic Procedure for the Assembly Line Balancing, *Naval Research Logistics Quarterly, 25,* 299–308.

Piper, J. P. and Zoltners, A. A. (1976) Some Easy Postoptimality Analysis for Zero–One Programming. *Management Sci., 22*, 759–765.
Pippenger, N. and Fischer, M. J. (1979) Relations Among Complexity Measures. *J. Assoc. Comp. Mach. 26*, 361–381.
Posner, M. E. and Guigner, M. (1978) The Collapsing 0–1 Knapsack Problem. *Math. Progr. 15*, 155–161.
Rardin, R. L. and Unger, V. E. (1978) Solving Fixed-Charge Network Problems with Group Theory-Based Penalties. *Naval Research Logistics Quarterly 23*, 67–84.
†Read, R. C. (ed.) (1972) *Graph Theory and Computing*. Academic Press, New York.
Rosenkrantz, D. J. Stearns, R. E. and Lewis II, P. M. (1977) An Analysis of Several Heuristics for the Traveling Salesman Problem. *SIAM J. Comput, 6*, 563–581.

†Saaty, T. L. (1970) *Optimization in Integers and Related Extremal Problems*. McGraw Hill, New York.
Sahni, S., (1977) General Techniques for Combinatorial Approximation, *Operations Research, 25*, 920–936.
Sahni, S. and Horowitz, E. (1978) Combinatorial Problems: Reducibility and Approximation, *Operations Research, 26*, 718–759.
†Salkin, H. M. (1975) *Integer Programming*, Addison Wesley, Reading, Mass.
Savage, S. L. (1976) Some Theoretical Implications of Local Optimization, *Math. Programming, 10*, 354–366.
Schoch, M. and Lyska, W. (1978–1) Kombinatorische Algorithmen zur Lösung Spezieller nichtlinearer 0–1 Optimierungsaufgaben. *Math. Operationsforsch. Stat. Ser. Opt., 9*, 9–20.
Schoch, M. und Lyska, W. (1978–2) Ein weiterer kombinatorischer Algorithmus für 0–1 Optimierungsaufgaben, *Math. Operationsforsch. Stat. Ser. Opt. 9*, 307–320.
Schulz, K. (1978) A cutting Plane Algorithm and a Column Generation Algorithm as a Dual Algorithms. 31–42.
Seeländer, J. (1978) Einige Bemerkungen zur Bestimmung des Schnittranges in der ganzzahligen linearen Optimierung. *Math. Operationsforsch. Stat. Ser. Opt. 9*, 321–334.
Shapiro, J. F. (1977) Sensitivity Analysis in Integer Programming. *Annals of Discrete Mathematics, 1*, 467–477.
Shih, W. (1979) A Branch-and-Bound Method for the Multi-Constraint Zero–One Knapsack Problem. *J. Opl. Res. Soc. Japan, 30*, 369–378.
Smith, T. H. C., Srinivasan, V. and Thompson, G. L. (1977) Computational Performance of the Three Subtour Elimination Algorithms for Solving Asymmetric Traveling Salesman Problem. *Annals of Discrete Mathematics, 1*, 494–506.
Smith, T. H. C. and Thompson, G. L. (1977) A LIFO Implicit Enumeration Search Algorithm for the Symmetric Traveling Salesman Problem using Held and Karp's 1–Tree Relaxation. *Annals of Discrete Mathematics, 1*, 479–493.
Soyster, A. L., Lev, B. and Slivka (1978) Zero–One Programming with Many Variables and Few Constraints, *EJOR 2*, 195–201.
Stein, D. M. (1978) An Asymptotic, Probabilistic Analysis of a Routing Problem. *Math. Operations Res., 3*, 89–101.
Suzuki, H. (1978) A Generalized Knapsack Problem with Variable Coefficients. *Math. Programming, 15*, 162–176.

†Taha, H. A. (1975) *Integer Programming Theory, Applications and Computation*. Academic Press, New York.
Talbot, F. B. and Patterson, J. H. (1978) An Efficient Integer Programming Algorithm with Network Cuts for Solving Resource-Constrained Scheduling Problems. *Management Sci., 24*, 1163–1174.
Tarjan, R. E. (1978) Complexity of Combinatorial Algorithms, *SIAM Review, 20*, 457–491.
Thomas, G. S. Jennings, J. C. and Abbott, P. (1978) A Blending Problem Using Integer Programming on-line. *Math. Progr. 9*, 30–42.

Tien, B. N. and Hu, T. C. (1977) Error Bounds and the Applicability of Greedy Solution to the Coin-Changing Problem. *Operations Research, 25*, 404–418.

†Traub, J. F., ed. (1976) *Algorithms and Complexity: Recent Results and New Directions*, Academic Press, London.

†White, W. W. (ed.) (1978) Computers and Mathematical Programming, *Proceedings of the Bicentennial Conference on Mathematical Programming*, Gaithersburg, Maryland, 1976. ACM SIGMAP.

Willis, R. J. and Hastings, N. A. J. (1976) Project Scheduling with Resource Constraints Using Branch-and-Bound methods. *Operational Research Quarterly, 27*, 341–350.

Wolsey, L. A. (1975) Faces for a Linear Inequality in 0–1 Variables. *Math. Progr. 8*, 165–178.

Wolsey, L. A. (1977a) Valid Inequalities, Covering Problems and Discrete Dynamic Problems. *Annals of Discrete Mathematics, 1*, 527–530.

Wolsey, L. A. (1977b) Valid Inequalities and Superadditivity for 0–1 Integer Programs. *Mathematics of Operations Research, 2*, 66–77.

Wong, C. K. (1976) A Combinatorial Optimization Problem Related to Data Set Allocation. *RAIRO, Inf. 10*, 83–96.

Yamamoto, Y. (1978) The Held–Karp Algorithm and Degree Constrained Minimum 1-Trees. *Math. Progr. 15*, 228–231.

Zanakis, S. H. (1977) Heuristic 0–1 Linear Programming: An Experimental Comparison of Tree Methods. *Management Sci., 24*, 91–100.

Zionts, S. (1977) Integer Linear Programming with Multiple Objectives. *Annals of Discrete Mathematics, 1*, 551–562.

Active set 59
Algorithm 204
  exact (see branch and bound)
  cutting plane 109, 213, 224, 241, 261–264
  dynamic programming 96–131, 263, 265, 268
  enumeration 33, 48–51, 129, 133, 147, 154, 205, 218, 220, 231, 234–243, 261, 267
  group theoretic 129–131
  Lawler–Bell 35–41
  shortest path 121–128
  heuristic 206, 212, 213, 220, 227, 230, 238–243, 263–268
  Lagrangean multiplier 207, 262–267
  learning procedure 206, 219, 236, 240
  Monte Carlo method 218–220
  neighbourhood search 204–212, 220, 238, 243, 266, 269
Alternative optima 64, 69, 112
Approximate solution 92
Ascendant 188
  immediate 188, 189, 198
Assignment problem 71, 207–208

Backtracking 142–148
Basic variable 86, 213
Benders decomposition 163–181, 262, 270
Bound, computation of lower 58, 59, 63, 65, 70–73, 80, 87, 130, 135, 143, 145, 204, 211, 228, 240, 267
  computation of upper 58, 63–66, 80, 86–91, 109, 113, 118–120, 143, 204–205, 212
Branch-and-bound 57–95, 129, 131, 204–206, 212, 213, 221–230, 240, 261–269
  filter method 182–203
  general framework of 59, 62, 88
  Land and Doig 57, 78
  principle 58, 62, 65, 69, 70, 78, 111
Branching variable 72, 73, 90

Cargo loading problem 18–19, 102
Complexity of algorithm 259, 266
Computing time 142, 207, 244, 260, 264
Congruence 93, 94, 213
Consequence 48, 142–147, 192–195
Constraint, active 214
  integrality 78, 81
  logical 19, 205, 262
  objective function 141, 144–149, 155–159, 192, 196, 216–219, 230–231, 236, 238
Cost 206–208, 221–225, 233, 244, 267, 268
Covering system 190, 191
Critical path 265
Cutting plane methods of Gomory (see algorithm)

Decision variable 98, 99
Decomposition rule 58–70
Descendant 48, 188–189, 195–199
Dominated solution 118, 120
Dual feasibility 87
Duality theorem 140, 168, 173, 184, 198
Dynamic programming (see also algorithm)
  optimality principle of 96, 97, 100, 102, 106, 124
  recursion 110, 121

Edge colouring of hypergraphs 265
Empty container problem 27, 28
Enumeration (see algorithm)
Expected number 219

Farkas' theorem 164, 165
Feasibility 144, 214
Filter 183, 194, 196, 198
Finite Abelian group 118, 122
Fixed charge problem 22–24
Flexible enumeration 236, 238
Flow-shop problem 260
Fourier–Motzkin elimination 214, 215
Function, approximation of 30, 31
  continuous 167
  convex 23
  monoton 35
  nonlinear separable 30
  piece-wise linear 210

Gomory cuts 92
Group decomposition 121
Group theoretic algorithm (see algorithm)

Hermitian Canonical form 214
Hermitian-cone problem 214–218
Heuristic algorithm (see algorithm)
Hierarchy graph 92

Incidence matrix 222
Infeasibility 14, 144, 147–149, 156, 197, 214
  of set 205
  measure 205–212, 232, 233, 238
Integer polyhedra 261
Integrality condition (constraint) 78, 81
Investment problem 24–26

Knapsack problem 15–18, 57, 64, 99, 264, 265
  multi- 265, 269

Lagrangean, multiplier 207, 267
  relaxation 262, 265
Land and Doig method 57
Lawler-Bell algorithm (see algorthm)
Learning procedure (see algorithm)
Lexicographic order 28, 33–35, 123
Linear combination 213, 216, 231, 236
Linear programming 87, 140, 182
Local search technique 204, 205, 210, 212, 220

Man—machine interaction 241, 262, 270
Mathematical programming 13
Matrix reduction 71
Measure 205, 206, 211, 212, 219, 220, 242, 262
Memory requirement 64
Minkowski's theorem 165
Monte Carlo method (see algorithm)
Multi-branch procedure 133

Neighbourhood, search (see algorithm)
  structure 204
Network 264, 265
NP-completeness of problem 259, 264
Numerical ordering 34, 36—39

Ordering 16
  lexicographical 28, 33–35, 123
  numerical 34, 36–39
  of coefficient 216
  of variable 204, 209, 261
  partial 33

Parametric solution 101, 105
Partial ordering (see ordering)
Penalty 88, 90, 92, 95
Permutation 69, 206–208
Plant location problem 26–27
Polyhedral cone 163, 165, 170, 171, 173
Polyhedron 22, 23, 31, 32, 163, 170, 171, 198
Predecessor 48
Prime implicant of a truth function 223
Probabilistic method 218, 259, 260
Problem, assignment 71, 207
  bounded 120, 121, 130, 131
  cargo loading 102
  investment 24–26
  knapsack 15–18, 57, 64, 99, 264, 265
  linear programming 163, 165, 171
  mixed integer programming 13, 32, 57, 78, 83, 86, 94, 166, 213, 262, 268
  multi-knapsack 265, 269
  multistage decision 96, 99
  of large size 63
  plant location 26—27
  pure integer 69, 166, 171
  quadratic assignment 267
  relaxed 63
  restricted assignment 207
  scheduling 260, 265, 266
  separable nonlinear 105
  set covering 204, 215, 256
  set partitioning 222–224, 267
  special structure of 57, 63
  travelling salesman 19, 57, 69, 73, 206, 207
  unbounded 119–131
  upper bounded 120

Quadratic assignment problem 267

Random choice 205
Relaxation method 212, 215
Restricted enumeration 234

Scheduling problem 260, 265, 266
Selection rules 88
Set, bounded 167
  closed 167
  compact 167
  convex 79, 80
  of feasible solution 65, 70, 97, 112
Set covering problem (see problem)
Set partitioning problem (see problem)
Shortest path 121, 123
Single-branch procedure 133, 182, 198
Single optimum function method 108, 118
Slack variable 108
Solution, approximate 92
  dominated 118, 120
  optimal 64, 66, 69, 71, 79, 110, 124, 213, 215, 222, 226, 227, 233, 244
  ordered 41–48, 155, 156, 186–199

parametric 101, 105
pseudo 41–48, 65, 142, 148, 186, 189, 191, 193–196, 200, 204, 220, 231–244
root of 148, 155, 192, 199
Solution, free 185, 188
pseudo 185–190, 195
Standard deviation 219
State variable 98, 99, 105
Stopping rule 121
Successor, function 206–209
structure 204, 205
Surrogate constraint 134–143, 147, 148, 182, 183, 198, 213, 216, 219, 231–233, 238, 242, 243, 265
strength of 134, 135, 138–142
Symmetricity 70

Test 64, 66, 70, 130, 142–149, 192–195, 197, 204, 220, 231, 236, 239, 242
Tightening method 216
Travelling salesman problem (see problem)
Triangle inequality 70
Truncated tree 190, 191
Truncation 189, 190, 195
Truth function 223

Unimodular, matrix 214
transformation 214
Upper bounding technique 109

Variable, basic 86
branching 72, 73, 90
candidate 155, 156
decision 98, 99
fixing of 144–147, 149, 155, 156, 158, 192, 195, 197
slack 108
state 98, 99, 105
tied 147

Weight of constraint 206, 208, 231, 233
Worst alternative 91

ISBN 963 05 2004 4