DEVELOPMENT AND APPLICATION OF NONLINEAR OPTIMIZATION TECHNIQUES

By

Gautam Mitra

Thesis submitted for the degree of Doctor of Philosophy
in the University of London.

Institute of Computer Science.

July 1968.

Abstract:

Some techniques of non-linear optimization have been developed and their application discussed. The Introduction, Chapter 1, contains a brief survey of the more important non-linear programming techniques and considers how these are related to each other. In Part 1, Chapters 2 and 3, a dynamic programming algorithm and a program for the optimum choice of the sites and heights of transmission towers are described. This is a self-contained case study.

In Part 2, Chapters 4, 5 and 6, the theory and development of two specific types of non-linear programming problem are outlined and further applications discussed. The problems discussed in this part are basically constrained in a linear space, but non-linearity is introduced when

1) some or all the variables defining the linear space are restricted to integer values, or

2) a non-linear (quadratic) objective function is considered.

The problems dealt with in Part 3, Chapters 7, 8, and 9, are non-linear because a combination of the conditions (1) and (2), above is considered. Further, in this part the relationship between mathematical programming and heuristic techniques is discussed. A substantial experiment with the tree-search algorithm for the solution of the fixed-charge problem is treated in detail and the significance of various branching heuristics tested. A new way of developing solution techniques is described whereby the algorithmic solution, i.e. the true solution, is obtained by a finite heuristic search.

A list of the types of problems which can be tackled by the techniques described in the thesis is given below.

1) Minimum cost sequential routing problems (Tower spotting).

2) Integer linear programs.

3) Convex quadratic programs (with parametric facility).

4) Integer quadratic programs (convex quadratic objective).

5) Linear transportation problems with fixed charges.

6) Linear and quadratic fixed charge problems.

The computer implementation and several techniques used in the dynamic programming work described in Part 1 are original. The work reported in Part 3, is again original and is claimed to be a contribution to the theory of mathematical and heuristic programming. In Part 2 some of the work done on integer linear programs, Chapter 4, is original. The convex quadratic program described in Chapter 5 is an implementation and extension of existing techniques. But the application of quadratic programming in road design, Chapter 6, is quite new. In Chapter 10 the conclusions reached in the earlier Chapters are appraised.

ACKNOWLEDGEMENTS

Much of the work reported in this thesis was carried out during the tenure of a Research Fellowship at the University of London Institute of Computer Science, and I first wish to thank the Director of the Institute, Professor R.A. Buckingham, for his encouragement and support throughout these three years.

Thanks are due to Mr K. Wolfenden for his general supervision, valuable criticism and help at various stages of research reported in this thesis.

In addition I would like to express my appreciation to all my colleagues at the Institute for many stimulating discussions, and especially to Professor D.E. Barton for his expert advice on statistical matters. To Professor E.M.L. Beale of Imperial College and CEIR Ltd. I am similarly indebted.

I would, further, like to thank Miss Sittika Ali for her great patience in typing this thesis. In the concluding stages of this thesis, the help rendered by Miss Mary Lee and Miss Sonia Lyons in correcting the English grammar and several awkward phrases has proved to be invaluable.

Finally, I must thank my mother who is my first tutor in mathematics; but for her moral support this thesis would have never been completed.

CONTENTS

		Page
NOTATION		1
Chapter 1	An Introduction to Nonlinear Optimization Problems and Methods for Their Solution.	3
PART 1	A Computer Technique for Optimizing the Sites and Heights of Transmission Towers - Case Study in Dynamic Programming.	19
Chapter 2	Optimum Tower Spotting-Formulation and Solution by Dynamic Programming Analysis.	23
Chapter 3	A Computer Program for Tower Spotting-Development, Use and Extensions.	41
References (PART 1)		63
PART 2	Nonlinear Mathematical Programs.	67
Chapter 4	An Improved Algorithm for the Solution of Integer Programs by the Solution of Associated Diophantine Equations.	68
Chapter 5	Convex Quadratic Programming by satisfying Kuhn-Tucker Condition and using a Symmetric Tableau	85
Chapter 6	Application of Quadratic Programming	112
References (PART 2)		128
PART 3	Tree Search Schemes and Heuristic Methods	131
Chapter 7	A Branch and Bound Algorithm for Mixed Integer Quadratic Programs.	135
Chapter 8	Dichotomizing Procedure for the Fixed Charge Transportation Problem.	151

		Page
Chapter 9	Dichotomizing Procedure for the Linear and Quadratic Fixed Charge Problems.	169
References (PART 3)		213
Chapter 10	Concluding Remarks and Discussion.	216
Appendix 1	B.E.S. Specifications (1960) for 400kV L.6. Lines	222
Appendix 2	Design Data for the L.6. Towers	226
Appendix 3	Duality Theory of Linear Programming and Primal (Dual) algorithm.	227
Appendix 4	Duality Theory of Quadratic Programming	230
Appendix 5	Rules for Pivotal Transformations on the Linear and Quadratic Programming Tableaus	231
Appendix 6	Listing of Tower Spotting (Data Processing) Program (EMA)	237
Appendix 7	Listing of Tower Spotting (Dynamic Programming) Program (EMA)	248
Appendix 8	Listing of Direct Search Method for Integer Programs (Algol)	270
Appendix 9	Listing of Symmetric Quadratic Program (Algol)	278
Appendix 10	Listing of Symmetric Quadratic Program-Branch and Bound (Algol)	288
Appendix 11	Listing of Fixed Charge Transportation Program/ Dichotomies (Algol)	300
Appendix 12	Listing of Fixed Charge Program/Dichotomies (Algol)	309

Page

Appendix 13 Input Format and Data for the Fixed Charge Problems and their Solutions 329

Appendix 14 Data for a Fixed Charge Problem Formulated as an Integer Quadratic Programming Problem 347

List of Supplementary Material:

1. Instruction Sheet for Digitalizing Transmission Line Route Data.

2. Profile Drawings (Three) for a Typical Section of Line Comparing the Design obtained by Manual Procedure with that of the Computer.

3. Specifications for Using the Symmetric Quadratic Program 1 Parametric - Technical Note TN4.

4. Publications and Reports:

   (a) A Computer Technique for Optimizing the Sites and Heights of Transmission Towers - A Dynamic Programming Approach.

   (b) An Improved Algorithm for the Solution of Integer Programs by the Solution of Associated Diophantine Equations.

   (c) Dichotomizing Procedure for the Fixed Charge Transportation Problem.

   (d) Dichotomizing Procedure for the Fixed Charge Problem.

Remarks on notation and the numbering of figures, equations, references

Figures are numbered such that the first number indicates the chapter, and the second, the number within the chapter (e.g., Fig. 2.3; Chapter 2, Figure 3). Only figures 1 and 2 in Part 1 of the thesis are numbered differently to get round the obvious difficulty. The equations are presented in round brackets, the first number refering to the chapter and the second to that within the chapter (e.g., (3.1); Chapter 3, equation 1). The references are presented in square brackets, the first number referring to the part of the thesis, and the second to the number within this part (e.g., [1.5]; Part 1 reference no. 5). These references are also quoted in the Introduction, Chapter 1, or Conclusions, Chapter 10, which do not fall within the three parts of the thesis.

Notation

All ... symbols and notations are defined in the body of the thesis, as and when these arise. For ... convenience of reference the more important ones are explained below.

$\mathcal{S} \equiv \{s_1, s_2, \ldots, s_n\}$ denotes a set of indices; the indices in the set may or may not be ordered.

$S1 \subset S$ means S1 is a subset of S.

$s_i \in S$ denotes membership of a set or space, i.e. s_i member of S.

The set (or logical) operators are replaced by their Algol representations:

$\cap \leftrightarrow$ and, $\cup \leftrightarrow$ or, $\neg \leftrightarrow$ not.

$x = (x_1, x_2, \ldots, x_n) \in \chi$ in E^n refers to an n-tuple, a point or a vector, in the n-dimensional euclidean space belonging to the set χ.

The gradient of a function,

$\nabla f(x)$, denotes $\left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right)$, the n-component vector of partial derivatives.

The Hessian of a twice differentiable function f(x),

H_f, denotes an n×n matrix, with elements given by $\frac{\partial^2 f(x)}{\partial x_i \partial x_j}$.

If a,b,c, are integers then a ≡ b mod(c) implies (a-b) is divisible by c. Thus the integrality requirement is often represented as a ≡ 0 mod(1), i.e., a must be an integer.

$[\alpha]$ denotes the mathematical integral part of a real number α, so that $\alpha = [\alpha] + f$, where $0 \leq f < 1$, f denoting the fractional part of α.

In the linear and quadratic programming problems considered, it has mostly been assumed that $x_1, x_2, \ldots, x_n$ are the structural variables and $y_1 = x_{n+1}, y_2 = x_{n+2}, \ldots, y_m = x_{n+m}$, the slack variables. Similarly, $u_1, u_2, \ldots u_n$ are the dual variables of the primal structural variables $x_1, x_2, \ldots, x_n$; $v_1 = u_{n+1}, v_2 = u_{n+2}, \ldots, v_m = u_{n+m}$ are the dual variables of the slack variables $y_1, y_2, \ldots y_m$. For all practical purposes, Lagrange multipliers and the dual variables are considered as synonymous terms.

Chapter 1

An Introduction to Nonlinear Optimization Problems and Methods for their Solution

a. General

The role of mathematics as an aid to the processes of scientific problem solving has been established for a long time. Until recently the power of mathematics to provide models, perfect or imperfect, for almost any physical situation has raised more questions than could be answered satisfactorily at that time. However, the rapid development of digital computers over the last two decades has greatly extended the applicability of mathematics, and it has become increasingly possible to obtain numerical solutions to such mathematical models and even to add to the refinement or the complexity of the models which can be solved.

Optimization, ranging as it does from the classical mathematical problem of maximizing or minimizing a function of one or more variables over a given domain to the design optimization of complex static systems or the performance optimization of dynamic systems, merits study in its own right. The twin tasks of formulating an optimization problem mathematically and developing a solution technique for it are equally challenging and have received the attention of many distinguished mathematicians, including Fermat, Gauss, Lagrange, Fourier, and more recently Kantorovich, Dantzig, Bellman, Kuhn and Tucker. Similar problems in the allied field of Game-Theory have been considered by Borel and von-Neumann.

Variational methods have been used for many years in the solution of optimization problems which arise in the study of physical systems. Recently, there has emerged another class of optimization problems arising in such fields as operations research and economics, and which require entirely new mathematical methods. In 1947 G.B. Dantzig and L. Hurwicz proposed the simplex method, the celebrated algorithm for the solution of linear

programs, and this marked the beginning of a new era in optimization studies. Since then linear programming and its extensions, remarkable for their model building and computational power, have seen very rapid development alongside that of the digital computer, and today these are among the most widely used techniques of applied mathematics.

Many optimization problems, especially those arising from real-life situations, are in some sense nonlinear (as defined in the next section) and the development of solution techniques for these naturally calls for a proper understanding and analysis of their underlying mathematical structure. The study of nonlinear optimization models has yet to develop into a rigorous methodology, and considerable effort is being expended at the present time to develop both the theory and application of such models. Many specialized procedures have been developed for these new problems under the headings of nonlinear programming and dynamic programming, as well as extensions of classical minimization techniques.

Nonlinear optimization models usually fall into two categories; models in which the objective function and the constraints are highly nonlinear, or those in which these are mildly nonlinear. The problems in the first category are best tackled by methods developed from the classical theory of extrema of functions of many variables, or by direct search methods, all of which consider the behaviour of the objective function as one of paramount importance. In mildly nonlinear problems, on the other hand, mathematical programming techniques, which devote more attention to the feasible region defined by the constraints and which are in some way related to the simplex search of linear programming, have been applied very successfully. Dynamic programming, in principle, should solve problems of both these categories. In practice, however, this recursive equation technique is most suited to discrete and or sequential optimization models. In this thesis only dynamic programming and nonlinear mathematical programming will be considered further.

In the rest of this chapter some theoretical points concerning the structure and the solution of non-linear optimization problems are considered. In Section b the general mathematical programming problem is defined, and non-linearity and the convexity property are considered in relation to both the objective function and the constraints. In Section c the Lagrange multiplier technique is introduced and consideration given to the Minkowski-Farkas lemma, duality and the Kuhn-Tucker conditions for saddle-point optima. A recent approach to the solution of constrained optimization problems by the use of penalty functions is described in Section d. Separable programming is introduced in Section e and Section f discusses the recurrence relations of dynamic programming and the reduction of dimensionality by the use of Lagrange multipliers. Section g considers the motivation for employing heuristic techniques to solve otherwise formidable problems. Section h indicates the dimensions of the various classes of problems which can be solved using present generation computers and established techniques.

b. <u>The general mathematical programming problem; nonlinear objective functions and constraints; convexity.</u>

The general mathematical programming problem may be defined as that of determining a vector $x = (x_1, x_2, \ldots x_n)$ in euclidean n-space E^n, which maximizes an objective function $f(x) = f(x_1, x_2, \ldots x_n)$ subject to the constraints

$$g_i(x) = g_i(x_1, x_2, \ldots x_n) \{ \leq \text{or} = \text{or} \geq \} b_i, \quad i = 1,2,\ldots m. \qquad (1.1)$$

In general, if these constraints include the non-negativity conditions

$$x_j \geq 0, \quad j = 1,2,\ldots n, \qquad (1.2)$$

it is convenient to consider them separately. Sometimes some or all of the variables x_j may be further restricted to integral values only. Convexity and linearity are two important properties which these functions might (or might not) possess.

A function $\phi(x)$ is linear in an open domain D if the gradient vector $\nabla\phi(x)$ exists and is invariant at every point of D. Functions for which this property does not hold are said to be non-linear. In particular, if either $f(x)$ or any of the $g_i(x)$ in the above statement of the general mathematical programming problem is non-linear, then the problem is likewise nonlinear. If some or all of the components of x are constrained to be integers, then again the problem is non-linear. In fact, if all the components of x must take integral values, then one is no longer concerned with euclidean n-space E^n but with a subspace of this, an integral domain which possesses special algebraic properties [1.3].

A function is said to be convex if its value is never underestimated by linear interpolation, a very cogent definition due to Beale [2.5]. Mathematically stated, a function $\phi(x)$ is said to be convex over a convex set X in E^n if for any two points x_1 and x_2 in X and for all λ, $0 \leq \lambda \leq 1$,

$$\phi[\lambda x_2 + (1-\lambda)\, x_1] \leq \lambda\, \phi(x_2) + (1-\lambda)\, \phi(x_1). \tag{1.3}$$

The convex set X may be all of E^n or, as is frequently the case, the non-negative orthant. It follows that the set

$$S = \{(x,z) : \phi(x) \leq z\} \tag{1.4}$$

is convex if $\phi(x)$ is convex. The function $\phi(x)$ is said to be strictly convex where the strict inequality (1.3) holds. Finally, $\phi(x)$ is said to be concave if $-\phi(x)$ is convex.

If $f(x)$ is twice differentiable at $x = \bar{x}$, then in the neighbourhood of $\bar{x}$ it may be approximated quadratically by the function $F(x)$, where

$$F(x) = f(\bar{x}) + \nabla f(\bar{x})\,(x-\bar{x}) + \tfrac{1}{2}\,(x-\bar{x})'\, H_f\, (x-\bar{x}). \tag{1.5}$$

In view of the importance of this quadratic approximation $F(x)$, the following

results should be noted.

1) The function F(x) is convex (or strictly convex) if the quadratic form associated with the Hessian H_f is positive semidefinite (or positive definite). A simple test for this property of definiteness is considered in Chapter 5, Section f.

2) If the Hessian H_f is positive semidefinite for all $\bar{x} \in E^n$, then f(x) is convex and admits of a global minimum.

3) Any unconstrained minimum of f(x) occurs at a point x* such that $\nabla f(x^*) = 0$.

4) The constrained optimization problem:
Minimize f(x) subject to the constraints

$$g_i(x) \leq b_i \ , \ (\ i = 1,2,\ldots m),$$

has a global minimum if f(x) is convex and the constraints generate a convex set.

c. <u>Lagrange multiplier technique, duality, Minkowski-Farkas lemma, and Kuhn-Tucker conditions.</u>

Consider the general mathematical programming problem:

Maximize f(x) subject to

$$g_i(x) \leq b_i \ , \ i = 1,2,\ldots m. \qquad (1.6)$$

The classical approach to the solution of this problem is to introduce an auxiliary function

$$L(x,v) = f(x) - v[g(x) - b] \qquad (1.7)$$

of n x-variables and m v-variables, the latter known as Lagrange multipliers, and then to investigate the stationary points of L(x,v). The function L(x,v) is said to possess a saddle point at $[x^*, v^*]$ in E^{n+m} if there exists

$\epsilon > 0$ such that for all x, $|x-x^*| < \epsilon$ and all v, $|v-v^*| < \epsilon$,

$$L(x,v^*) \leq L(x^*,v^*) \leq L(x^*,v). \tag{1.8}$$

A fundamental property concerning a matrix and its transpose, and constituting the basis of the solution of all linear inequality systems, is the theorem of the 'Separating Hyperplane'. This states that for any m×n matrix A and any m-vector b, either

$$Ax = b, \tag{1.9}$$

has a solution $x \geq 0$, or

$$A'y \geq 0 \ , \ b'y < 0 \tag{1.10}$$

has a solution y, but not both. This theorem is equivalent to the Minkowski -Farkas lemma upon which the duality theory of linear and non-linear programming is primarily based. Wolfe [1.11] has considered duality in relation to the general mathematical programming problem. In terms of the maximization problem considered in the last section, Wolfe's duality theory may be stated as follows:

For continuous and differentiable functions f(x) and $g_i(x)$, such that f(x) is concave and $g_i(x)$ convex, the two problems:

(i) Maximize f(x) subject to

$$g_i(x) \leq 0 \ , \quad i = 1,2,\ldots m,$$

(ii) Minimize $f(x) - \sum_{i=1}^{m} u_i \, g_i(x)$ subject to

$$\nabla f(x) = \sum_{i=1}^{m} u_i \, \nabla g_i(x) \ , \ u_i \geq 0$$

are dual to each other, and the extrema of these two problems, if both exist, will be equal. The special cases of duality in linear and quadratic programming are considered in Appendices 3 and 4 respectively.

Necessary conditions for the existence of a local optimum solution to the general constrained optimization problem are given by the famous theorem

of Kuhn and Tucker. The n-vector $x^* = (x_1^*, x_2^* \ldots x_n^*)$ maximizes $f(x)$ subject to $g_i(x) \leq b_i$, $i = 1,2,\ldots m$, if and only if, there exist non-negative Lagrange multipliers $v_1, v_2, \ldots v_m$ such that

$$\frac{\partial f(x^*)}{\partial x_j} - \sum_{i=1}^{m} v_i \frac{\partial g_i(x^*)}{\partial x_j} = 0, \quad j = 1,2,\ldots n \tag{1.11}$$

$$\left.\begin{aligned} g_i(x^*) &\leq b_i && (1.12)\\ v_i &\geq 0 && (1.13)\\ \sum_{i=1}^{m} v_i(g_i(x^*) - b_i) &= 0 && (1.14) \end{aligned}\right\} \quad i = 1,2,\ldots m \,.$$

If one now adds the nonnegativity constraints $x_j \geq 0$, or $-x_j \leq 0$, $j = 1,2,\ldots n$, then, associated with these, Lagrange multipliers $u_1, u_2, \ldots, u_n$ may be introduced, which leads to the further conditions

$$\left.\begin{aligned} \sum_{j=1}^{n} x_j^* u_j &= 0 && (1.15)\\ u_j &\geq 0 && (1.16) \end{aligned}\right\} \quad j = 1,2,\ldots n$$

at the same time modifying (1.11) to

$$\frac{\partial f(x^*)}{\partial x_j} - \sum_{i=1}^{m} v_i \frac{\partial g_i(x^*)}{\partial x_j} + u_j = 0, \quad j = 1,2,\ldots n. \tag{1.17}$$

In general, certain conditions (which the constraints are required to satisfy and known as the 'constraint qualification') impose severe restrictions on the set of feasible solutions in the vicinity of x^*. In the quadratic programming problem of maximizing a quadratic form subject to linear constraints (Chapter 5) 'constraint qualification' is automatically satisfied. The majority of the convex quadratic programming methods work by satisfying the conditions (1.12) ... (1.17) while employing some simplex type search procedure.

From the standpoint of computation the saddle-point property is often of great assistance. For instance, the convergence of the differential

gradient method of Arrow and Hurwicz [1.7] on the Lagrange multiplier technique for the reduction of dimensionality in dynamic programming, is primarily due to this property of the Lagrangean function considered.

d. Sequential unconstrained minimization techniques.

No review of nonlinear optimization techniques can be complete without mentioning the recently developed penalty function methods for the solution of constrained optimization problems. This method, known by the above name or as 'the created response surface technique', involves solving a sequence of unconstrained minimization problems. The most appealing feature of this approach is that it can be implemented trivially by making use of one of the many algorithms for unconstrained optimization ([1.15], [1.16]). The original work in this field is due to Carrol [1.12].

Let G_i be a nonincreasing function of the constraint function $g_i(x)$ and be defined for $g_i(x) < 0$, and let $g_i(x)$ be convex. Consider the problem of maximizing the concave function $f(x)$, subject to the constraints $g_i(x) \leq 0$, $i = 1,2,\ldots m$. Construct the modified objective function

$$F(x,r) = f(x) + r \sum_{i=1}^{m} G_i\,(g_i(x)), \qquad (1.18)$$

where r is a perturbation parameter and the G_i's are known as penalty functions. Fiacco and McCormick [1.13] have proved the following relationship between the original and the constructed problems;

$$\lim_{r^k \to 0} \max_x F(x,r^k) = \lim_{r^k \to 0} \max\Big[f(x) + r^k \sum_{i=1}^{m} \frac{1}{g_i(x)}\Big] = \max_{x \in R} f(x) \qquad (1.19)$$

where $G_i(g_i(x)) = \dfrac{1}{g_i(x)}$, and R denotes the convex solution space. If the G_i functions assume the form $G_i(g_i(x)) = \ln(-g_i(x))$ or $G_i(g_i(x)) = \dfrac{1}{g_i(x)}$ (note that the penalty functions $G_i \to -\infty$ for $g_i(x) \to 0^-$) then G_i's become concave for all x such that $g_i(x) < 0$. Under such conditions $F(x,r^k)$ admits of global concavity and can be maximized by a suitable unconstrained maximization technique; therefore, to solve the constrained optimization

problem it is only necessary to solve a sequence of unconstrained problems.

Work is currently going on in the field to find out ([1.5])

1) the suitability of various alternative G_i functions,

2) the conditioning of the Hessian matrix of the function $F(x,r)$,

3) suitable ways of determining the successive values of r and the convergence criteria.

e. Solution techniques for the variables separable problems [3.7], [1.8], [1.9]

In situations where the general nonlinear functions, either the constraint functions $g_i(x)$ or the objective function $f(x)$, can be expressed as the sum of one or more nonlinear functions of a single argument, it is possible to reformulate the nonlinear problem and solve it with the aid of a linear programming code. This is more or less the essence of separable programming due to Miller [3.7, pp 89-100], and is probably one of the most promising nonlinear programming technique to date.

The assumptions made above imply that mathematically the following representations of $f(x)$ and $g_i(x)$ are possible,

$$f(x) = \phi_{01}(x_1) + \phi_{02}(x_2) + \ldots + \phi_{on}(x_n) \tag{1.20}$$

$$g_i(x) = \phi_{i1}(x_1) + \phi_{i2}(x_2) + \ldots + \phi_{in}(x_n). \tag{1.21}$$

Consider any one of the above functions of single argument, say $\phi_{is}(x_s)$. To linearize this function let some nonnegative variables, called special variables λ_k, be introduced which may be considered as weights attached to a few discrete points P_k taken on this function (see the accompanying Figure). Let (a_k, b_k) denote the coordinates of the point P_k and let there be q such points. The situation can be approximated by having a convexity row

$$\lambda_1 + \lambda_2 + \ldots + \lambda_q = 1, \tag{1.22}$$

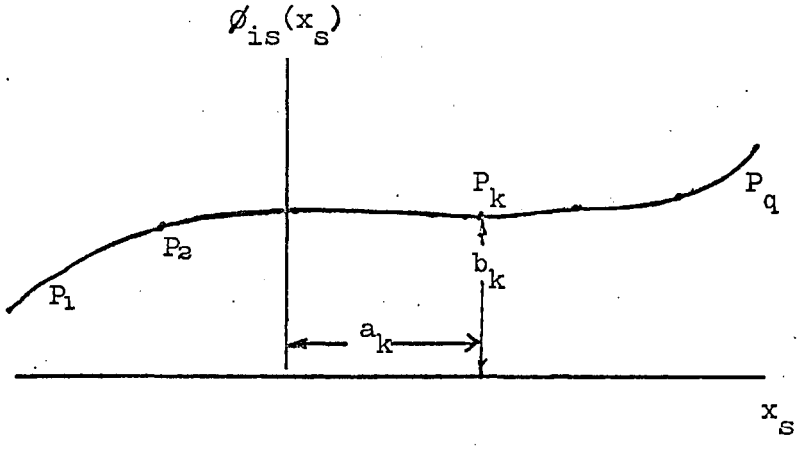
and a reference row

$$a_1\lambda_1 + a_2\lambda_2 + \ldots + a_q\lambda_q = x_s, \tag{1.23}$$

and a representation of the function value $\phi_{is}(x_s)$,

$$b_1\lambda_1 + b_2\lambda_2 + \ldots + b_q\lambda_q = \phi_{is}(x_s) \tag{1.24}$$

In a convex problem the above linearization overestimates the value of the function. The convexity of all the ϕ_{ij}'s in a convex problem ensures



global optimality - when a linear programming code may be applied without any modification. For non-convex problems Miller showed that the rules for linear programming can be trivially modified to allow only two adjacent lambdas to be in the basis at any one time. This implies that a linear approximation of the functions between any two adjacent points be considered, and ensures convergence to a local optimum solution. The following

important features in connection with this approach may be noted.

1) If a function is not explicitly variable separable and contains product terms then this can be handled by either taking logarithm of such terms or by expressing these as the difference of the square terms of two new variables [1.8], [2.5].

2) In a non-convex problem, the first optimal solution may be obtained by neglecting the additional restrictions on the λs mentioned above; one therefore obtains the solution to a problem only considering its convex hull. The special restrictions can be brought in subsequently to obtain a local optimum solution which is globally important [1.19].

3) Employing an interpolation scheme in conjunction with this method one can obtain a more refined solution than that obtained by the initial discrete approximation to the function [1.9].

4) If one adds to the problem the facility of integer programming one can obtain the global optimum [1.9] for non-convex problem.

f. Dynamic programming; a sequential multistage decision technique.

Dynamic programming stems from a fundamental postulate of Bellman [1.17] known as the principle of optimality. On the basis of this it is possible to deduce certain recurrence relations which often lead to useful computational schemes for the solution of both continuous and discrete type optimization problems.

For example, consider an objective function of the variables separable type as in Section e

$$f(x) = \phi_1(x_1) + \phi_2(x_2) + \ldots + \phi_n(x_n). \tag{1.25}$$

For this the following property holds,

$$\begin{aligned} \max_{x_1, x_2, \ldots, x_n} f(x) &= \max_{x_1, x_2, \ldots, x_n} [\phi_1(x_1) + \phi_2(x_2) + \ldots + \phi_n(x_n)] \\ &= \max_{x_n} \phi_n(x_n) + \max_{x_1, x_2, \ldots, x_{n-1}} [\phi_1(x_1) + \phi_2(x_2) + \ldots + \phi_{n-1}(x_{n-1})] \end{aligned} \tag{1.26}$$

where the problem considered is one of unconstrained maximization. When the maximizing problem is constrained in some way the above concept may be applied with the added proviso that the maximizing solution must belong to the region R defined by these constraints. Consider the problem of maximizing (1.25) subject to,

$$\begin{aligned} &a_{11} x_1 + a_{12} x_2 + \ldots + a_{1n} x_n \leq \alpha, \\ &a_{21} x_1 + a_{22} x_2 + \ldots + a_{2n} x_n \leq \beta, \\ &x_1, x_2, \ldots, x_n \geq 0 \text{ and integer, } a_{1j}, a_{2j} > 0, \alpha, \beta > 0. \end{aligned} \tag{1.27}$$

A sequence of maximizing problems and the corresponding maximizing functions $F_k(\gamma, \delta)$ may be defined, where the quantities γ, δ, the arguments of these functions, must lie within the ranges

$$\begin{aligned} &0 \leq \gamma \leq \alpha, \\ &0 \leq \delta \leq \beta. \end{aligned} \tag{1.28}$$

This sequence of functions is connected by the recurrence relation

$$F_k(\gamma, \delta) = \max_{0 \leq x_k \leq \min([\frac{\gamma}{a_{1k}}], [\frac{\delta}{a_{2k}}])} [\phi_k(x_k) + F_{k-1}(\gamma - a_{1k} x_k, \delta - a_{2k} x_k)], \quad k = 2, 3, \ldots, n \tag{1.29}$$

and

$$F_1(\gamma, \delta) = \max_{0 \leq x_1 \leq \min([\frac{\gamma}{a_{11}}], [\frac{\delta}{a_{21}}])} \phi_1(x_1). \tag{1.30}$$

When stated in full the function $F_k(\gamma, \delta)$ stands for the maximum of $\sum_{j=1}^{k} \phi_j(x_j)$ subject to the conditions

$$\begin{aligned} &\sum_{j=1}^{k} a_{ij} x_j \leq \gamma \\ \text{and} \quad &\sum_{j=1}^{k} a_{2j} x_j \leq \delta. \end{aligned} \tag{1.31}$$

In connection with determining the maximum of $F_k(\gamma, \delta)$ over x_k (1.29), it may be noted that $\hat{x}_k(\gamma, \delta)$, the value of x_k which maximizes this function, is itself dependent on γ and δ and may not be unique. If at any stage k more than

one value of x_k yields $F_k(\gamma,\delta)$, one can tabulate each $\hat{x}_k(\gamma,\delta)$, which allows one to obtain the multiple optima if these are required to be computed. If k denotes the stage in this process of computing this sequence of maximizing functions, then $F_k(\gamma,\delta)$ is the maximum if only the stages 1,2,...k existed; the quantities (γ,δ) on the right hand side of the constraints (1.31) represent the amounts which may be utilized at the current stage. The physical interpretation of the equation (1.29) is that an optimal policy for k stages must have the property that whatever the choice of the decision variable for the kth stage, the policy must be optimal for the remaining k-1 stages for this choice of x_k. This is the interpretation of Bellman's principle of optimality as applied to this problem, often called the resource allocation problem.

In general, this version of dynamic programming is called a discrete (a finite integrally spaced grid for γ,δ, x_j is defined), deterministic (because of the model), multistage (because of the sequence of maximizing functions), decision technique. From the view-point of computation it may be noted that,

1) by a direct search the sequence of functions $F_k(\gamma,\delta)$, $\hat{x}_k(\gamma,\delta)$ against γ,δ are compiled to produce a 3-dimensional table, this dimensionality growing with increasing number of state variables, that is constraints,

2) this search procedure is more economical than a direct enumeration technique [1.18].

In problems in which continuous variations of the values of x_j are allowed, the above technique may be extended by considering a discrete grid of γ,δ and x_j so that a table similar to that in the integer case may be compiled. A point in favour of this approach is that the global optimum solution, depending upon the grid size of γ,δ, and x_j, is obtained without having to impose the concavity property on the relevant functions. However, if the functions

$\phi_j(x_j)$ are concave, the dimensionality of the tables which need to be computed may be reduced by the introduction of Lagrange multipliers as follows.

If from some a-priori considerations, it may be concluded that $\sum_{j=1}^{n} a_{ij} x_j \leq \beta$ is the constraint which must be satisfied as exact equality at the optimum, then for fixed numerical values of u the maximizing function to be considered is

$$Z(x) = \sum_{j=1}^{n} (\phi_j(x_j) + u a_{ij} x_j) \; ; \qquad (1.32)$$

the maximization of which has to be carried out subject to the single constraint $\sum_{j=1}^{n} a_{ij} x_j \leq \alpha$, requiring only a two dimensional table to be compiled. Having obtained this maximum, if it turns out that the optimal solution x^* does not satisfy the other constraint, i.e. $\sum_{j=1}^{n} a_{ij} x_j^* \neq \beta$, the value of u is re-adjusted and the procedure repeated. Under the assumptions outlined above the method can be shown to converge.

Dynamic programming, it may be emphasized, is a method of analysis aimed towards devising suitable computational techniques for many different types of problems. A class of combinatorial problems taken from graph theory, for instance the shortest route problem and the travelling salesman problem ([1.20]), may be solved by this approach with varying degree of success. A variant of the shortest route problem, a sequential routing problem with special types of constraints, has been formulated and solved (chapters 2,3) by this technique.

g. Motivation and the application of heuristic methods

In the fields of cybernetics and artificial intelligence in particular, attempts have been made to program a computer to solve problems involving some elements of optimization in a human like fashion. This approach towards problem solving requires taking decisions and proceeding further with the solution in the light of the information already acquired from the results of decisions taken at earlier stages. In problem areas where the exact solution of a problem is impossible or where known computational

schemes for its solution cannot be implemented on present day computers, due to limitations of storage or performance, good compromise solutions have been obtained by the use of so called heuristic techniques. For example consider the two combinatorial problems: assembly line balancing([1.21]) and the travelling salesman problem([1.4],[1.20],[3.2]). The integer programming formulation of the first problem is a particularly difficult one to solve, whereas Tonge[1.22] has developed a powerful heuristic method for its solution. The travelling salesman problem similarly admits of integer programming formulation([1.4]), or may be tackled by a dynamic programming algorithm ([1.20]). However, the branch and bound method of solution which appears to be the most promising method to date, can produce a good solution quickly and, computing time permitting, can also determine the global optimum. Further consideration is given to this general approach in Part 3 of this thesis.

h. The state of the art in the computer realization of the various mathematical programming techniques.

In the last ten to fifteen years, there has been a considerable increase in the size of the programming problems which can be solved and this progress can be ascribed to,

a) improvements in the algorithms developed,

b) advances in computer technology.

Since reliable and tested figures are rather hard to come by, one cannot but be less specific in discussing the state of the art regarding the type and the size of the problems which can be solved by existing computer codes. However, the figures presented below may be taken as a guide to the magnitude of the problems which may be solved using known mathematical programming techniques on large present generation computers such as the CDC6600 and the UNIVAC1108.

Linear Programming: Upto 20,000 constraints and almost unlimited number of of variables. For problems such as the transportation problem which are solved by special techniques the corresponding figures will be much higher.

Integer Programming: Upto 1000 integer variables and constraints. The ability to solve integer and mixed integer problems bears no relation to the size of these problems. Even quite small problems may remain unsolved because of their inherent difficulty.

Quadratic Programming: Between 500 and 1000 variables and as many constraints.

Travelling Salesman Problem: 50 to 60 city problems may be tackled for exact solutions. For larger problems suboptimal solutions may be obtained employing heuristic techniques.

Constrained or unconstrained minimization of functions which are highly nonlinear but continuous and differentiable: About 80 to 100 variable problems have been solved by the various function minimization routines.

PART 1

A Computer Technique for Optimizing the Sites and Heights of Transmission Towers - A Case Study in Dynamic Programming

Introduction

Conceptually, and also from the view point of computation, the solution of a class of problems by dynamic programming may be looked upon as equivalent to finding minimal paths in networks. In a network that exhibits some kind of sequential property, time sequential or space sequential, the principle of optimality [1.17] can be very elegantly illustrated. In this introduction some important features of such networks are considered, and the associated dynamic programming problems together with their applications are briefly reviewed.

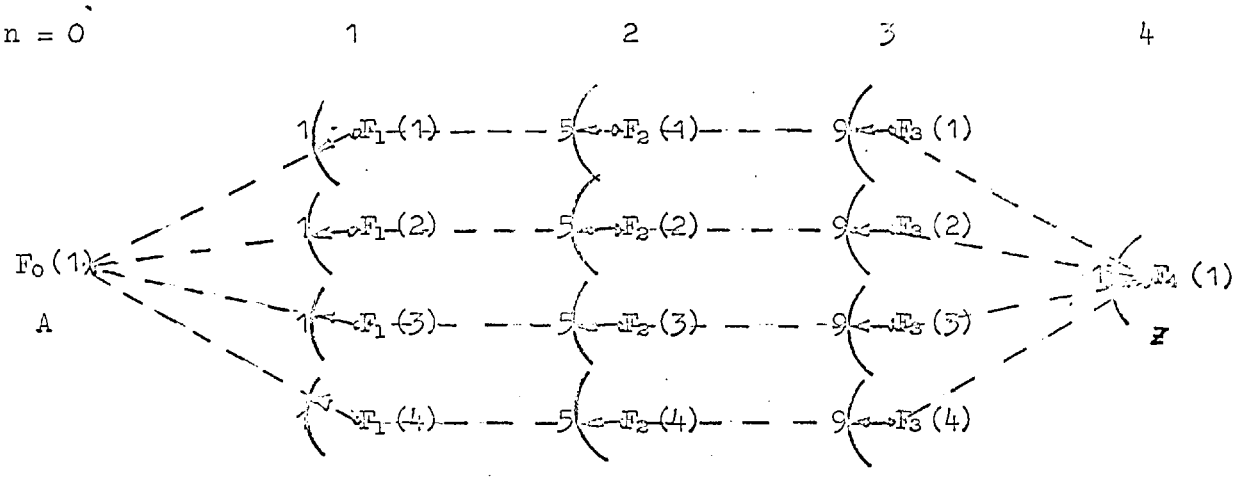
Consider the network shown in Fig. 1 ; let with every arc, that is the line connecting any two nodes, a distance or some relevant cost figure be associated. Let n= 0,1,2,3,4, denote the stages of some route or some process, and j= 1,2,3,4 the states at some intermediate stage i= 1,2,3, the initial and final stages being constrained to be in one state (j=1) only. The connections of the network depend on the nature of this model, but one assumption that will hold in general is that there are no connections between the nodes contained in any particular stage. In this network consider the problem of moving from the initial node A, stage 0, state 1, to the node Z, stage 4, state 1, following a minimal path. This may be looked upon as a problem embedded in the family of problems of finding the minimal paths, say $F_i(j)$, i= 1,2,3, ; j=1,2,3,4; to all the intermediate nodes, and finally obtaining the solution $F_4(1)$; $F_0(1)$ may be defined to be the value of such a minimizing function at the root node. The network is said to possess a sequential property if one is allowed to move only from

a stage of lower index number to a higher one. The connection may take place only between the successive stages as in the signal flow graph formulation of the Markov-process [1.37] or over many stages as is the case in the application considered in this thesis. An algorithm to solve the above minimal path problem may be simply stated as follows. At stage 1, no matter what one is going to do in the remaining stages one must arrive at the various nodes (states) 1,2,3,4 in this stage optimally, from the previous stage. For stage 2, the same argument may be repeated, this time arrival from stage 0 and also from the optimal solutions of stage 1 are considered. One again obtains the minimizing solution $F_2(j)$ associated with each node (state) of this stage 2. The process is continued until optimal solutions to each of the nodes including the terminal node Z are produced and by doing this one develops the solution tree shown in Fig. 2 rooted in the starting node $F_0(1) = A$. A mathematical statement of this inductive algorithm is not given here, it will be considered in Chapter 2. An arc in this tree represents a decision, a path, a policy, and since the minimization is carried out over successive stages this approach is known as the discrete deterministic (or probabilistic) multi-stage decision technique. It is well known that when terminal states are fixed dynamic programming may be applied by considering backward or forward induction, hence one could obtain an alternative tree rooted in Z by starting at stage 3 and working backward to node A.

Using a dynamic programming approach, Gortsko [1.33] has solved the problem of locating factories in the appropriate range (0-s) along a railway, P. Dale [1.34] likewise has solved the problem of re-aligning underground railway tunnels. The French Road Board [1.36] has adopted a dynamic programming approach to obtain alignment of highways and Jefferis and Fegley [1.35] have used similar techniques for the laying of pipelines at minimal cost. These problems may all be looked upon as minimal path problems in a finite connected network. When the multi-stage problem is not deterministic, as in the Markov process, it may be

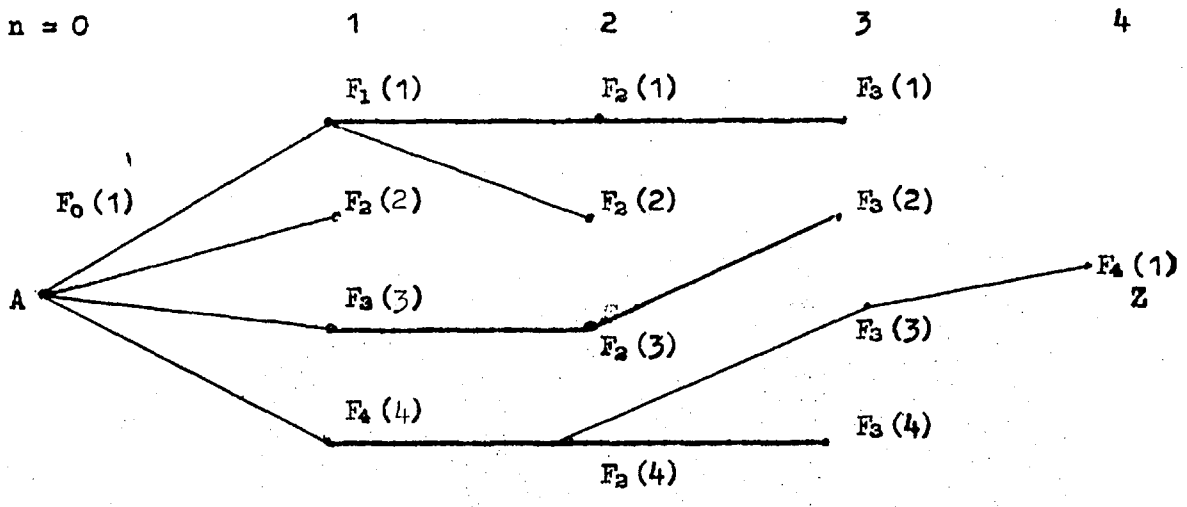
said that the value iteration and the policy iteration approach for the solution of such Markov processes with reward also admit of network interpretation.

The tower spotting problem belongs to this class of deterministic sequential routing problems in networks. In Chapter 2 of the thesis, this engineering design problem is formulated as a non-linear programming problem. A dynamic programming algorithm for the solution of this problem is then described. In Chapter 3 the computer implementation of the method, processing of the input data and other pertinent feature of the algorithm and the program are considered.



The numbers associated with a node indicate the number of arcs connected from the left.

Fig. 1.1



A hypothetical solution tree.

Fig. 2

Chapter 2

Optimum Tower Spotting - Formulation and Solution by Dynamic Programming Analysis.

Introduction

Given the survey data along a transmission line route and the choice of available towers of suspension type, and angle towers, the problem of choosing the sites and heights of towers satisfying the statutory specifications is a design engineering problem ([1.39]). In Section a of this chapter the notation and the design constraints are introduced, in Section b some computational details concerning sag calculation, base setting level, swing clearance, etc are elaborated. Section c briefly describes the manual design procedure, and Section d reviews the computer methods developed by other workers for the solution of this problem. The non-linear constrained problem is formulated as a minimization problem in Section e, where the application of the dynamic programming algorithm for its solution is also outlined. The design features of the problem, and the algorithmic aspect of the solution technique are kept independent of each other as far as possible.

a. Notation and description of the design constraints

For ease of reference to the notation, the symbols used in the rest of the chapter are first defined; where appropriate these are also illustrated in Fig. 2.1, 2.2, 2.3, and in Appendix 1.

List of symbols

A_{hot} the parameter of the parabola in which the line for the span under consideration is assumed to hang at 122° F, the statutory design temperature for computing clearance,

A_{cold} the same under ice and wind loading and at 22° F,

x the horizontal distance from the beginning of the line section,

$s_1(x)$ the maximum single span limit at x. (This is a function of x,

since on high grounds exposed to wind, long single spans may be prohibited etc.),

s_2 the maximum double span limit,

X_{test} an ordered set of test sites,

X_{tower} an ordered set of tower sites,

$X_{clearance}$ an ordered set of clearance sites,

$v(x)$ the elevation of the lowest conductor of the line above sea level,

$v_i = v(x_i)$ the same at a support situated at x_i,

$Z(x)$ the statutory clearance from the ground or any other object at x,

$u_{1i}(x_{1i}, s_{1i})$ the elevation of the centre line of the terrain at the ith tower base, test site X_{1i} and having a base setting gradient of s_{1i},

$c(x,h)$ the cost of erecting a tower of height h at the site $x = X_{1i} = x_i$,

$u^e_{-1}(x_p, -d_p)$, $u^e_0(x_p, 0)$, $u^e_1(x_p, d_p)$, the elevations of three points, a point to the left of the centre line offset by $-d_p$, a point on the centre line, a point to the right of the centre line offset by d_p,

$x_{i+1,i}$ the distance from x_i to the lowest point of the span S_{i+1},

$x_{-1,i}$ the distance from x_i to the lowest point of the span S_i,

S_i the ith span, i.e. the distance between the tower sites x_{i-1} and x_i.

T_{22}^{+} the maximum allowable design tension (. ' in lbs) of the line at 22° F with ice and wind loading,

T_{122} the tension of the line at the temperature of 122° F at which clearance is to be calculated,

Seq the equivalent span of a line section given by the relationship $Seq=\left(\frac{\sum s_i^3}{\sum s_i}\right)^{\frac{1}{2}}$,

A the equivalent cross section say in inch^2 of the conductor,

E the Young's modulus(lb/inch^2)of the composite conductor,

W the weight per foot run of the conductor,

α the coefficient of linear expansion of the conductor,

{Note A,E,W, and α are given as equivalent values for the composite ACSR (Aluminium Cored Steel Reinforced) conductor}

q loading factor, the ratio of the loading due to the weight of the conductor plus ice and wind to the loading due to the weight of the conductor alone.

The problem of finding a line layout consists of choosing an ordered set of tower sites, and an integer M defining the number of sites in this set,

$$X_{tower}= \{x_i \mid i= 1,2,\ldots M, \text{ and } x_{i-1} < x_i\}. \qquad (2.1)$$

($x_0 = 0$ the initial tower site and x_M= total distance;

the initial and final tower sites are fixed)

from an ordered set of test sites

$$X_{test}= \{X_j \mid j= 1,2,\ldots N, \text{ and } X_{j-1} < X_j\}, \qquad (2.2)$$

where $X_{tower} \subset X_{test}$, and choosing a . ' ' tower height h_i for each tower site x_i, the tower heights being chosen from the set of available

towers, (i.e. $h_i \in$ Htower), where

$$\text{Htower} = \{H_k \mid k = 1,2,\ldots Q, \text{ and } H_{k-1} < H_k\}, \tag{2.3}$$

Q being the number of alternative tower heights available. In (2.2) for $N \to \infty$, the problem admits continuous variation in the choice of the tower sites. For feasibility, the line layout must satisfy the design specifications (BES.16 specifications for the case considered Appendix 1 and 2) which consist of;

(I) a maximum single span constraint

$$S_i = (x_i - x_{i-1}) \leq s_1(x_i) \quad ; \; i=1,2,\ldots M, \tag{2.4}$$

(II) a maximum double span constraint

$$S_i + S_{i+1} = (x_{i+1} - x_{i-1}) \leq s_2 \quad ; \; i = 1,2,\ldots M-1, \tag{2.5}$$

(III) an uplift constraint which sets a limit on the possible angle of swing of the line under transverse wind loading

$$x_{1,i} + x_{-1,i} \geq W(x_{i+1} - x_{i-1}) \quad , \; i = 1,2,\ldots M-1; \tag{2.6}$$

in design language W is half the minimum ratio of weight span $(x_{1,i} + x_{-1,i})$ to wind span $(x_{i+1} - x_{i-1})/2$,

(IV) and finally the clearance constraints, whereby the sag of the line at 122° F is first calculated and then the distance of the conductor from the side slope Fig. A 1.2 of an offending structure offset to the line (Fig. A 1.3, Fig. A1.4 (a)) computed and checked against the clearances specified. The set

$$\chi_{\text{clearance}} = \{\chi_p \mid p = 1,2,\ldots R, \; \chi_{p-1} < \chi_p\} \tag{2.7}$$

defines the locations at which one tests for statutory clearance; the corresponding elevations $u^c_{-1}(\chi_p, -d_p)$, $u^c_0(\chi_p, 0)$, $u^c_1(\chi_p, d_p)$, are used in computing these clearances. In (2.7) again, if $R \to \infty$ the clearance is

evaluated continuously over the length of the line. The clearance computation involves base setting of the towers, the sag calculation and the geometrical configuration of the line when subjected to side wind. These features are considered in the next section.

b. Sag and clearance computation

A chain freely hanging between two supports P and Q, Fig. 2.3, takes up the shape of a catenary, if there exists no shear force acting perpendicular to the length of the conductor [1.39]; this therefore should be the curve defining any individual span of the line. However, the assumption concerning the shear force does not strictly hold for the line conductors (this view was upheld by the designers in C.E.G.B) and it is claimed [1.38] that a parabola provides a better approximation to such a curve. Nothing further will be said about the error introduced by such an approximation.

The tension of the line is at its maximum when the line is at the lowest ambient temperature, 22° F, is loaded with ice and experiences a side-wind. The tension at 122° F is lower and is given by the cubic equation (2.13), ([1.39], [1.38]), which can be deduced from the first principles by considering the elastic extension of the line length and the extension due to the change in temperature. The two tensions T_{22}^{+} and T_{122} therefore determine two curves, the cold curve and the hot curve, the former being used for the purpose of obtaining the design tension and the weight span, and the latter for the purpose of clearance. The normal design practice consists of the following steps:-

(I) Obtain the parameter of the parabola under ice and wind loading A_{old}, by the equations, (2.8), (2.9)

loading factor

$$q = \frac{\sqrt{(\text{weight of ice in lb/ft} + W)^2 + (\text{wind load lb/ft})^2}}{W} \qquad (2.8)$$

$$= \frac{\sqrt{(1{\cdot}0104 + 1{\cdot}125)^2 + 1{\cdot}416^2}}{1{\cdot}125} = 2{\cdot}277,$$

and,

$$A_{cold} = \frac{q \cdot W}{2T_{cold}} \text{ length}^{-1} \qquad (2.9)$$

$$= \frac{2{\cdot}277 \times 1{\cdot}125}{2 \times 12000} = (1{\cdot}0675 \times 10^{-4})\ \text{ft}^{-1},$$

the numerical values being those for the B.E.S.16 line under consideration.

(II) From the general equation governing the shape of the parabola between any two supports x_i and x_{i+1},

$$v(x) = A_{cold}(x - x_i)(x - x_{i+1}) + \frac{v_{i+1} - v_i}{x_{i+1} - x_i}(x - x_i) + v_i \qquad (2.10)$$

(where $x_i \leq x \leq x_{i+1}$ $v_i = v(x_i)$, $v_{i+1} = v(x_{i+1})$ the elevations of the line conductor at the ith and $i+1^{th}$ supports, compute the distances to the minimum points from the point of intermediate support v_i at x_i,

$$x_{1,i} = \frac{v_i - v_{i+1}}{2A_{cold}(x_{i+1} - x_i)} + \frac{x_{i+1} - x_i}{2} \qquad (2.11)$$

$$x_{-1,i} = (x_i - x_{i-1}) - \left\{ \frac{v_{i-1} - v_i}{2A_{cold}(x_i - x_{i-1})} + \frac{x_i - x_{i-1}}{2} \right\} \qquad (2.12)$$

and hence check the uplift constraint (2.6) considered in the last section.

(III) Obtain the tension of the line for an equivalent span Seq (the need for introducing this is discussed in the following paragraph) at a maximum ambient temperature of 122°F, given

by the equation,

$$t.\alpha.E.A = \frac{(W.q)^2.Seq^2.E.A}{24(T_{22}^{+})^3} - T_{22}^{+} = \frac{W^2.Seq^2.E.A}{24(T_{122})^2} - T_{122} \qquad (2.13)$$

where, t=(122 - 22) denotes the change in temperature and the other symbols are as explained in the notation. For known values of the other parameters this reduces to a cubic equation in T_{122}, and is trivially solved. From T_{122} obtain the parameter of the hot curve.

$$A_{hot} = \frac{W}{2T_{122}} \text{ length}^{-1} \qquad (2.14)$$

and compute the elevation of any point of the line conductor by the equation,

$$v(x) = A_{hot}(x - x_i)(x - x_{i+1}) + \frac{v_{i+1} - v_i}{x_{i+1} - x_i}(x - x_i) + v_i. \qquad (2.15)$$

On a terrain with no side slopes and no offending structures offset to the line it must be ensured that

$$v(x) - u_o^c(x,0) > Z(x) , \qquad (2.16)$$

$$x \in \chi\text{clearance}$$

so that the clearance specifications are satisfied.

Equation (2.13) can only be derived by considering any particular span length of the line; in other words starting from the design tension of T_{22}^{+}, the derived tension T_{122} is a function of this span length. To compute the true tension in each span, when all the spans of a line section are considered, a set of simultaneous cubic equations for the tensions of each individual span has to be solved. This the traditional manual method (from which the design specifications have emerged) is incapable of handling. Hence, design specifications postulate that for the purpose of checking clearance, one must obtain the tension corresponding to an equivalent span.

$$Seq = \sqrt{\frac{\sum_{i=1}^{M} S_i^3}{\sum_{i=1}^{M} S_i}} \qquad (2.17)$$

The cubic terms are introduced in this averaging equation since (2.13) is a cubic equation. It may be noted that the use of equivalent span underestimates the sag in a span shorter than this and overestimates the sag in a longer span.

The inequality (2.16) is not the only condition to be satisfied for all the clearance requirements. Two special cases are to be further considered and are outlined in brief.

Side slope

On grounds with side slopes one may assume that the terrain profile in side elevation is specified by two straight lines joining u_0^c to u_{-1}^c, and u_0^c to u_1^c. It is now necessary to compute the clearance of the line when it swings through an angle of θ° with the vertical direction, θ being the angle of the steepest rising side slope (Fig. A 1.2, Appendix 1), R_s the radius of swing of the line is computed by first obtaining $v(x)$ from the equation (2.15) and then subtracting this from the elevation of the hinge point. Finally, an inequality of the type (2.16) is tested for specified clearance.

Offending structures

For a structure offset to the line, clearance is computed, for an angle of swing θ, $\theta \leq 45$, by considering the line joining the nearest point on this structure to the hinge point, cf Fig. A 1.3, Appendix 1. The radius of swing R_s is obtained as above and one tests for an inequality of the type (2.16). For $\theta > 45^\circ$ the horizontal displacement is checked for the purpose of clearance, where θ denotes the angle made by the line joining the highest point on the structure and the hinge point (Fig. A 1.4, Appendix 1). The uplift constraint (2.6) ensures that the line can swing upto a maximum of 45°.

The elevation of the hinge point, Appendix 1, plays an important role in computing these swings. This point is obtained by linearly

interpolating between the points of suspension at adjacent towers and considering it at an offset distance of lc from the centre line, where lc denotes the length of the lowest cross arm of any tower.

Finally, it must be mentioned that on terrains with side slopes the conductor elevation at a support is given by the equation

$$v_i = u_{li} + H_k - g_{li} \times W_k \ ,$$

where u_{li} is the terrain elevation (of the centre line) at the li^{th} test site, i^{th} tower base $X_{li} = x_i$, g_{li} the steepest downward gradient, k the index of the tower and W_k its base width. The quantity $g_{li} \times W_k$ is the amount by which the base of the tower is set lower than it would be on flat ground; the geometrical configuration is illustrated in Fig. 2.5.

c. Manual design procedure

Computation of the operating tension and checking the uplift and the clearance constraints all form an integral part of the manual design procedure. The digital computation of the appropriate equations and the inequalities over the whole range of a line section is not feasible at the human speed of numerical processing. Hence an analogue method has been in use until recently at most of the design establishments. This method involves going through the following steps of design procedure.

(I) The coefficient of the parabola A_{cold} at the maximum design tension $T_{23}^{\uparrow}$ under ice and wind load is calculated (2.9).

(II) For a number of possible equivalent spans (say, Seq= 800,900...1600 ft, for B.E.S.I6 lines), the corresponding tensions of the hot sagging curve at 122° F are calculated (2.13); and from T_{122}(Seq), the coefficients of the parabola A_{hot}(Seq) are calculated.

(III) From (I) and (II), the cold curves and the hot curves for all the equivalent spans are plotted and plastic templates are cut to these shapes. One may further use clearance curves, which are obtained by displacing in the vertical direction each point of the hot curve by an amount equal to the specified ground clearance. Lastly, base setting curves for the tower base may also be used [1.39], (Fig. 2.4.)

(IV) In any given straight line section of the line, an equivalent span is chosen and the towers are sited; 'the location of supports should be such that a lateral adjustment of 10-15 ft in either direction may be permitted without a resulting increase in tower height. The most desirable allocation of supports is that which results in the least overall cost while keeping within the requirement of the specifications', [1.39].

(V) Having obtained a tower schedule, the actual equivalent span of the line, (2.17) is computed. If this is greater than that used in the first instance, then it follows that the clearance conditions will be satisfied in this case as well. However, if it is otherwise, then the template of the hot curve of the next higher equivalent span is used; and if clearance is violated towers are resited. The method is known to converge.

(VI) In doubtful cases, swing clearance, uplift constraints etc, are checked by actual numerical computation.

(VII) Where the line route changes direction OA, AB, Fig. 2.6, angle towers are chosen out of the available range , such that the angles of turn β_1, β_2 are less

than those of the angle towers used.

In the absence of a rigorous mathematical formulation of the objective function and a method of minimizing it, the cost of the solutions obtained by this method depend entirely upon the skill of the tower spotting personnel. In general the following rules are observed to obtain economical solutions:

1. Ensure that the sag templates fit the ground profile as closely as possible.

2. Select tower locations to utilize to the fullest extent the wind span capabilities of the various towers.

3. Keep the height of the angle towers as low as possible, since these are of higher cost than the suspension towers of same height, (Appendix 2).

It has been reported by various workers who have comparative figures for the manual and computer solutions, that this method proposes conservative designs. Since a visual/analogue approach is used to scan the terrain it might transpire that clearance is violated at some point on the line. In one of the comparative studies in which the author was involved this was known to have happened.

d. Computer methods adopted by other workers

Computer methods reported or proposed by other workers were studied in some detail. In all the case studies reported, savings in cost of both the solution proposed and its derivation over the traditional manual method have been claimed. These works may be conveniently divided into two main categories.

To the first category belong those methods [1.25], [1.27], [1.28],

[1.31] which are machine implementations of the manual design procedure. The computing capability is utilized to study many more alternative solutions than is manually possible and hence more economical solutions can be obtained by these methods. Converti, Hyland and Tickle [1.27] first developed a successful program along these lines and drew the attention of design engineers to the use of such techniques. They report a saving of 10 percent in capital cost compared with the manual method, recommend that aerial survey data should be suitably processed for input and that output should be oriented towards order placement and material control. Popp, Dabekis and Fullerton [1.28] have developed a very similar method. However, they have gone further to consider explicitly the various possible combinations of spans and tower heights, so that the most economical combination over a few spans may be obtained and the process repeated. Conceptually such an effort is equivalent to a direct enumeration of the subproblem of siting the towers and choosing the heights over a short length optimally. Direct enumeration cannot be done over the whole length simply because the number of combinations to be considered (which is equal to the number of tower heights raised to the power of number of test sites) is prohibitive. Bartelink [1.25] has also followed up this approach, and has incorporated a different feature - evaluating the cost of the distance traversed by various towers; the distance traversed being defined as the length between the lowest points of the adjacent spans. It seems American practice does not involve the use of equivalent span to check clearance, and in none of these three papers this point is considered. Hoare and Morwood [1.31] developed a computer program suited for Australian design specifications; for the purpose of optimization the technique of Popp et al has been adopted.

The works of Ahlberg and Palm [1.24] Shulman [1.30] Stagg and Watson [1.32] belong to the second category where the problem is rigorously formulated as a minimization exercise. Method of carrying out the minimization of the objective function, subject to the design constraints

are considered in these papers. That the recursive equation technique of dynamic programming may be applied to solve this problem was first suggested by Shulman [1.30]; he also pointed out the equivalence of the problem to that of routing in networks. However, the sequential property of this network was not specially mentioned by him. He does not report any computer implementation of the method and his formulation of the constraints is not rigorous. Ranyard [1.29], working under Wolfenden implemented the suggestions of Shulman and reported a successful run on very limited test data. Without explicit mention of the term dynamic programming, Ahlborg and Palm [1.24] in Sweden developed a program, which uses the same inductive procedure for obtaining the optimal solution. These authors have dealt with the basic design problems adequately and comprehensively. Further, considering the size of the machine for which the program was written, a Facit EDB with 2K core store and 8K drum store, the results reported by the authors compare very favourably with those obtained on present generation machines. The problems of processing the input data, loading of the line under different conditions, equivalent span computation etc are considered in sufficient detail and savings of cost are also reported. Stagg and Watson [1.32] have employed essentially the same dynamic programming approach as that due to Shulman. They also report saving in capital cost; however, computing experience seems to be limited.

e. Formulation and solution by dynamic programming

The constraints of the design were presented in Section a; to treat the problem as a design optimization exercise it is now necessary to consider the objective function.

$$C = \sum_{i=1}^{M} C(x_i, h_i) \qquad (2.18)$$

$$= \sum_{i=1}^{M} C(h_i), \text{ (if only the tower costs are taken into account)}$$

which must be minimized subject to the constraints, (2.4), (2.5), (2.6), and the clearance constraints, a simplified form of which is given in (2.16). The problem is non-linear since the constraints (2.6) and (2.16) are non-linear. Let j= 1,2,...N, the indices of the test sites X_j be identified as the stages of the process, let H_k denote the state, and let the connection of one tower height at a particular stage to another in a previous stage denote a decision. In the dynamic programming terminology a sequence of minimizing functions may be defined,

$$F_j(X_j,H_k),\ j= 1,2,...N,\ \text{and}\ k= 1,2,...Q \qquad (2.19)$$

denoting the cost (including the cost $C(H_k)$ of the current tower) of constructing a line from the stage 0, $X_0 = 0$ and a specified tower height h_0, (initial state), to the tower of height H_k at site X_j of the stage j, following an optimal policy. Conceptually this is equivalent to associating minimal cost paths to all the nodes defined by the stages and the states of a finite connected network. Fig. 2.7, shows what the network should actually look like in practice. However, it suffices to consider the equivalent rectangular grid illustrated in Fig. 2.8 and with reference to this the recursive equation of dynamic programming may be stated as

$$F_s(X_s,H_t) = C(H_t) + \min F_j(X_j,H_k),\ j=0,1,...s-1,\ k= 1,2,...Q. \qquad (2.20)$$

The rule for assigning the optimal cost value $F_s(X_s,H_t)$ to the node (s,t) is to search for a linkage $(s,t) \rightarrow (j,k)$ satisfying the constraints of the problem and which minimizes the expression on the right hand side of (2.20). This now specifies the optimal policy upto (s,t). Applying the recurrence relation (2.20) repeatedly over the range t=1,2,...Q, for s=1,2,...N successively [for s=0, t is specified and $F_0(X_0,H_t)=C(H_t)$ may be defined] a family of solutions is generated through to the end of the line section. As is usual with computational procedures employing the recurrence relation of dynamic programming, having generated the

family of solutions it only remains to find out the optimal solution at the last stage, $\min_k F_N(X_N,H_k)$, k= 1,2,...Q, and then trace back those linkages and the corresponding tower sites and tower heights, x_i, h_i, i= M-1, M-2,...1, ($x_i \in X_{tower}$, $h_i \in H_{tower}$) which constitute the optimal solution.

In deducing the above algorithm only one state variable, the tower height has been postulated. However, corresponding to each of the constraints (2.4), (2.5), (2.6) one could have associated a state variable. Such a formulation would lead to a 5-dimensional table of minimizing functions, as opposed to the 2-dimensional table which needs to be compiled following the present formulation. The computational burden involved for such a scheme will be neither economical nor feasible in current generation computers.

In place of the range of values for j shown in equation (2.20), j the previous test site index may need only cycle over the range,

$$j= I, I+1,...P \qquad (2.21)$$

where I is such that $X_s - X_I \leq$ single span limit, $X_s - X_{I-1} >$ single span limit, and P is equal to s-1. Finally, for the purpose of back tracking, and to simplify checking of the constraints, when a connection from a subsequent test site to the current one is being tried, it is necessary to store a vector of information associated with each minimizing solution, $F_s(X_s, H_t)$. Such a vector may consist of the optimal cost $F_s(X_s,H_t)$, previous site index j, previous height index k, and the distance to the minimum point of the span to the left measured from the current tower at the test site X_s. Further details regarding the computer implementation of the method are considered in the next chapter.

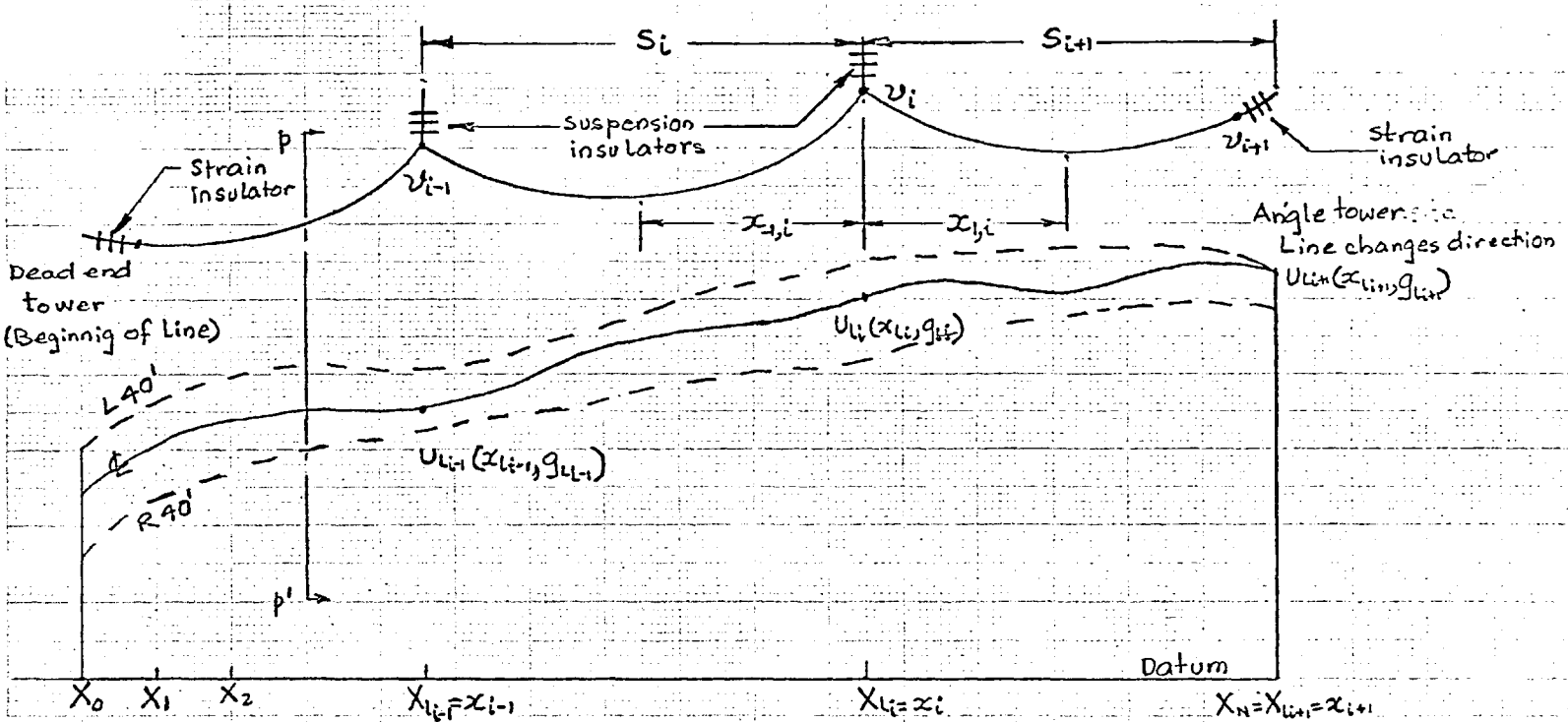


Fig.2.1

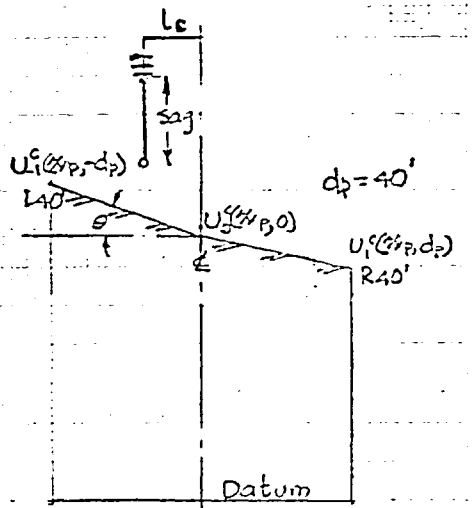


Fig.2.2.

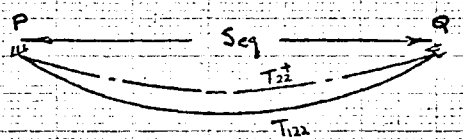


Fig.2.3.

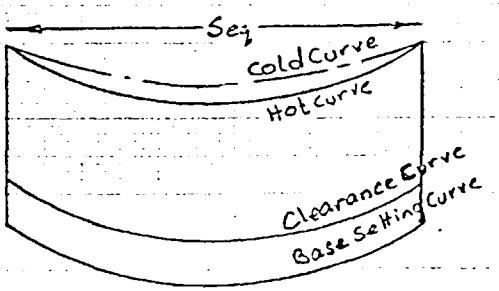
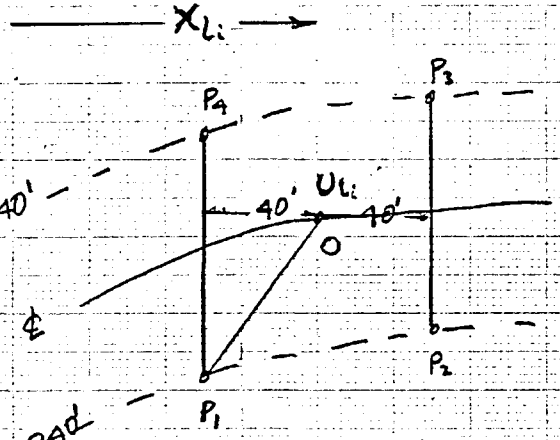
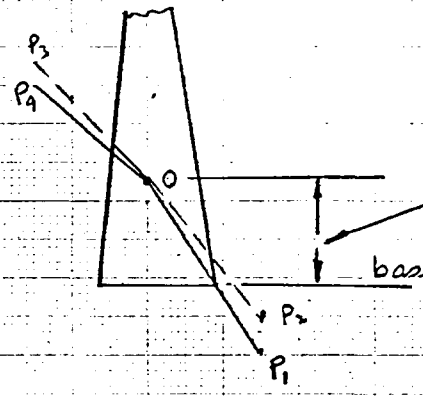


Fig.2.4.



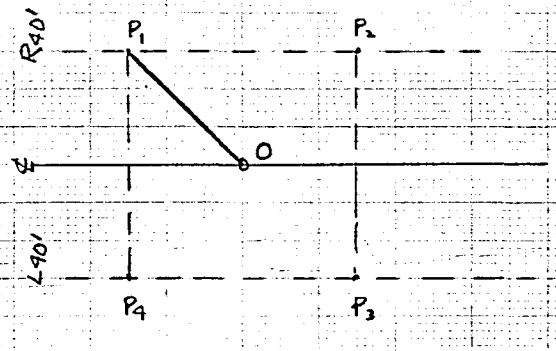
Elevation



Side view.

OP₁ direction of largest slope

q_{Li} = (U_{Li} − elevation of P_1)/40'√2



PLan

Base setting for towers to be sited at

O, X_{Li}, in a ground with side slope.

Fig-2.5.

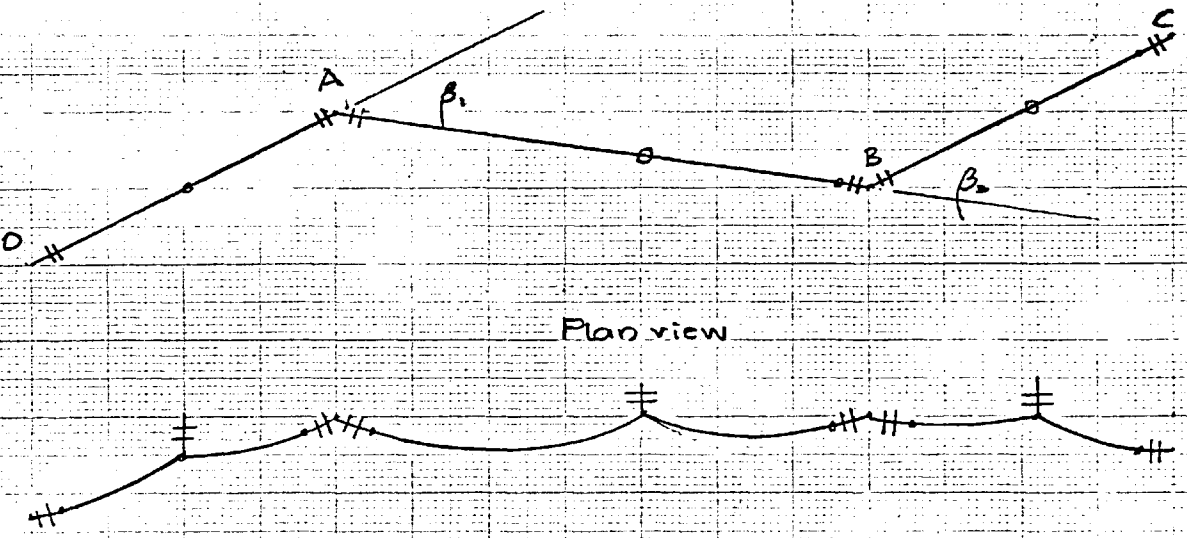


Fig.2.6.

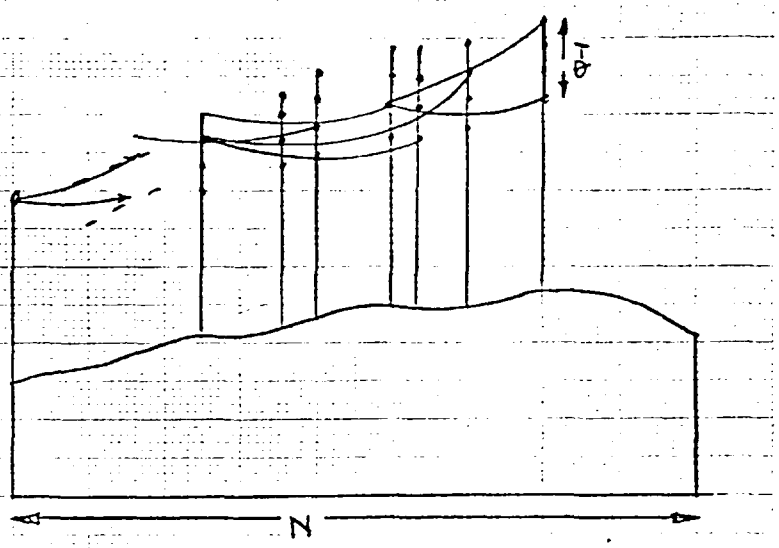


Fig.2.7

Two dimensional grid representation of test sites and tower heights

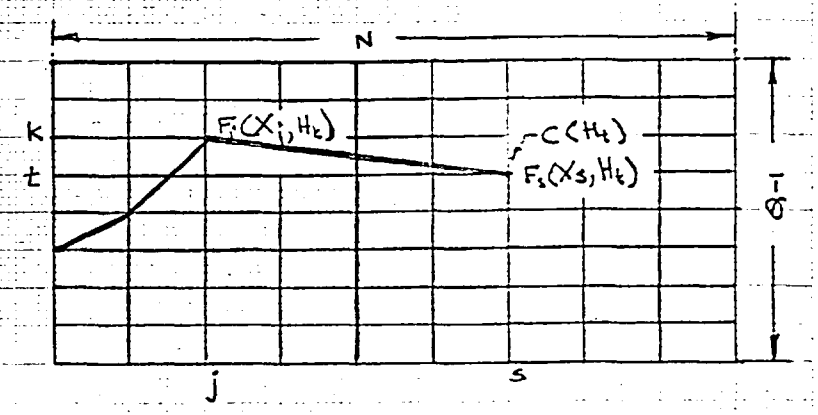


Fig.2.8.

Chapter 3

A Computer Program for Tower Spotting - Development, Use and Extensions

From the viewpoint of computer implementation the Tower Spotting problem splits up naturally into two sections, a data processing section which organizes the raw input data and writes it up on a magnetic tape, and the main program which applies the dynamic programming algorithm on this processed information to generate the family of minimal solutions as outlined in the last chapter. In Section a of this chapter the data processing program and its relevant features are described. Section b outlines the overall flow of control and some details of storage and back-tracking of the solution. Further, in this section some useful concepts involved in computing and checking the design constraints are presented. Section c discusses the results derived in an actual design exercise, the sensitivity of the solution and the computing time in relation to the size of the grid used. Section d suggests further works and extensions to the program.

a. The data processing program

A flexible data processing program constitutes an integral part of any worthwhile design automation (optimization) program. For instance, one of the applications [1.34] cited earlier was undertaken in view of the fact that the survey data was already available in a suitably processed form. This point has also been emphasized by other workers who have developed similar tower spotting programs. The survey data, as obtained by open-trek or aerial survey, is normally used to produce the profile drawings along the line route; the various straight line sections and the locations of the angle towers having already been decided (Fig. 2.6) prior to tower spotting. The survey data of the line route is therefore arranged conveniently as separate records for each straight line section. There are, at most, five items of information which need to be recorded at

any locations on the line route; these are chainage, elevations at the centre line, left forty feet offset, right forty feet offset and the amount of statutory clearance necessary at this location on the terrain. Ideally, such information should be captured at the stereo plotting stage from the aerial photographs. However, in the case study considered, this data was digitalized from the profile drawings. The instruction sheets enclosed (supplementary material) outline the rules specified for the guidance of the personnel who abstract this data. The information concerning each section of the line route is therefore represented as one record in terms of a card file, or, when processed, one fixed length record of a magnetic tape file.

Card file

Every card in the card file must be in three-numeric or five-numeric format, that is, it will contain either three or five numbers. Normally at each location, five numerical values are recorded as explained above, however, for terrains with no side slope it is only necessary to record chainage, elevation of the centre line, and the statutory clearance. To define the various sets X clearance, X test, (Chapter 2, Section a) and also to record supplementary information concerning the line section, say section number, angle of turn, span limits etc., special character markers are punched in the first column of each card, thus identifying different card types in the file (Fig. 3.1). The '*' character indicates a card whose information refers to a location on the route used both as a test site and a clearance site, the '-' character indicates a card relating to a clearance site only. Such cards arranged in ascending order of chainage form the bulk of each section record (Fig. 3.2). When an offending structure offset to the line needs to be represented one uses a card of type '-'. Further, the first number A_1 is set equal to 1 or 2, to indicate left or right offset, A_2 is set equal to d_p indicating the amount of horizontal offset, A_3 is set equal to u_p indicating the elevation of the offending point (Fig. 3.2, Fig. A 1.3, Appendix 1), and the other two

numbers A_4, A_5 have their usual interpretation of clearance and chainage. At the end of each section record there will be cards of two additional types. These are '+' cards detailing the elevations just before and after the finishing angle tower of the line, so that the tower base gradient for the angle tower (note this is sited skewed to the direction of the line) may be calculated. Finally the card with character marker '>' is included which details the line type, angle of turn etc.

On flat ground with no side slope, a card of type '(' will appear in the record where the flat ground starts, indicating that the following cards are of three numeric format; a card of type ')' is used to signal the change back to five numeric-format. Fig. 3.2 illustrates a record in terms of the card file of a hypothetical section of a line, and cards of all the above types are included for the purpose of exposition.

Magnetic tape file

In the magnetic tape file seven blocks are assigned to each section record, which comprises test site, clearance site and line-detail sub-records. A test site subrecord occupies the first two blocks and contains 2N alternate entries of chainage and elevation at the centre line of the route. The maximum gradient (Fig. 2.5) for the purpose of base setting is evaluated by an interpolation routine. The chainage of the test sites will have integer values; the gradients are scaled down so that these will be decimal fractions less than 0·1. The fractional values of the gradients are then tagged on to the integral values of the chainage. A typical entry,

chainage	elevation
325·017	200·67

indicates that at chainage 325 (=integer part of 325·017), the elevation of a point at centre line is 200·67 ft and the downward gradient is 0·17. If this were an upward gradient the first decimal figure would be '9' instead of '0'. Another and slightly different interpolation routine obtains the

base setting gradient at the finishing angle tower. This value is also used in the next section record and printed out after the current section record is processed. For instance, if it becomes necessary to update one section record, this gradient value at a starting angle tower, along with the relevant section number and block number on the magnetic tape all need to be read in.

The clearance site subrecord comes after the test site and occupies the next four blocks and 500 words of the seventh. The clearance subrecords are arranged exactly as in the card file, that is, five words per location. Subsequently, to check clearance in the main body of the programme, on terrains with only side slope (no offending structure) chainage, elevation at the centre line and cos θ (where θ is the angle of inclination of the higher of the two offset points with a point on the centre line (Fig. A 1.2)), need to be considered. For offending structures a special marker is set up and other relevant information concerning offset and elevation is considered. Finally, the last twelve words of the seventh block are designed to contain details of the section, e.g. section number, angle of turn, etc.

The use of a fixed-length record on magnetic tape immediately sets an upper limit on the length of a section which can be processed and stored in this way. Alternatively, for a given length of a section, this sets a limit on the fineness of the grid which may be used. For a mean grid length of, say, fifty feet (which is the finest one may wish to use) the upppr limit of the length of a section becomes 25,000 feet ≃ five miles. In practice, (this was subsequently explained by the designers in the CEGB tower spotting group) a tension tower must be sited after three drum lengths of conductor. Thus no section can be longer than 13500 feet, a drum length being 4,500 feet, and this is well within the limit.

To realize the layout and the manipulation of the data, a data processing program was written in EMA and is given in Appendix 6.

The program inputs the card file, interpolates the base cetting gradients and writes up the magnetic tape file. It also includes such simple but useful checks as cards being out of order, excessive side or longitudinal slope, etc.

It is perhaps appropriate to mention in this section that EMA does not possess a suitable data structure whereby the files described could be easily defined, input and processed. It is now widely recognized that for any worthwhile information processing application the programming language used must possess a suitable data structure and some facility for its flexible manipulation. Such facilities are only to be found in commercial languages like ACL, COBOL; however, other programming languages are also being designed in the light of these requirements.

b. Computer implementation of the dynamic programming algorithm

In its core the main program implements the recursive equation (2.20), checks the constraints for feasibility and stores the minimal solutions thus generated. The data transfer, computation and decision processes are illustrated in the flow chart Fig. 3.3 and its legends.

Starting from the initial dead end tower at the beginning of the line route, the section records consisting mostly of X test, X clearance, and the corresponding elevations for each straight line section are brought down in sequence from the magnetic tape into the main store. The clearance records are then transformed into elevation and inclination parameters, u_p and θ (see last section). With the details of X test and X clearance laid out in the main store, the computation progresses by unit increments in s(loop 1) and t(loop 2), the indices of the current node, and the search for the preceding connecting node is carried out by adding unit increments to j(loop 3) and k(loop 4), the preceding tower site and height indices. As explained in (2.21) j cycles over the range j=I(1)P, where I is determined by the maximum single span constraint (Section e Chapter 2). P is

set equal to s-1, but to reduce computing effort a minimum economic single span may be postulated such that

$$X_s - X_p > \quad \text{minimum single span,}$$

$$X_s - X_{p+1} \leq \quad \text{minimum single span,}$$

for the 16 lines considered the minimum single span is taken as 500 feet. When the linkages to the initial tower site ($X_j = 0$) are being tried, provision is made to ignore the double span and the uplift constraints. Normally, however, the solution corresponding to the preceding tower site (test site= X_j) and tower height k is unpacked (the way this information is stored/packed is considered later in this section) and the cost of the linkage computed. Only if this is less than the current best solution, are all the constraints tested to establish the feasibility or otherwise of this linkage.

To check the feasibility of a linkage and to record the cost of the corresponding solution, it is fundamental to evaluate the elements of a vector $Y=(y_1, y_2, y_3, y_4, y_5)$ where

y_1 denotes the elevation of the conductor at the preceding tower site (test site= X_j),

y_2 denotes the elevation of the conductor at the current tower site (test site= X_s),

y_3 denotes the elevation of the point of suspension of the insulator string at the preceding tower site,

y_4 denotes the elevation of the point of suspension of the insulator string at the current tower site,

y_5 the cost including the cost of the current tower, of building this line up to the current tower site (test site=X_s) and tower height, t.

From the Chart 3.1 and Fig. 3.4 it may be observed that four possible combinations of the points of suspension of the insulator and conductor elevations at the supports are feasible. On a section of short length it may be possible to connect the line from angle tower to angle tower by a single span. Chart 3.1 illustrates the method of synthesizing arithmetical expressions from the boolean variables b_1 and b_2, indicating the type of previous tower and current tower, i.e. whether these are of suspension or angle type. Use of such expressions avoids writing long texts of program with conditional transfer of control. In fact C.E.G.B. specified that the program should provide the facility of joining lines of different types, viz, L2 with L6 or vice versa. L2 type towers are different from L6 towers in height and string length; which meant $2^4 = 16$ possible combinations of the above type needed to be considered. In the actual program therefore, in the same way as explained in chart 3.1, arithmetical expressions involving four boolean variables were written to synthesize the relevant combinations.

Having obtained the vector γ, sag at 122° F may be computed trivially using the equations (2.14), (2.15); the tensions corresponding to the different span lengths at this elevated temperature are contained in the program in a tabulated form. The hinge point at any clearance site is obtained by linearly interpolating between y_3 and y_4.

Every time a feasible solution is obtained, the solution or rather the four items of information defining it, is stored temporarily. These four items may be represented as,

B_1 denoting the index of the preceding test site, j,

B_2 denoting the index k of the tower at this site j,

B_3 denoting the cost of building this line to the current node including the cost of the current tower,

B_4 denoting the distance to the minimum measured from the tower at the current test site.

Until j has cycled through the full range, j= I(1)P, new links are tested first for cheaper solution and then for feasibility, a cheaper feasible solution always overwriting the current best solution. Finally at the end of the cycle, the temporarily stored solution is transferred to the main store, this solution being of course associated with the node s, t. B_1, B_2 and B_3 being simple positive integers, are packed into one ward (one Atlas word contains 48 bits) each occupying 12 bits (this is the field used by the EMA 'pack' instruction). B_4 is a real number, may be positive or negative and is stored separately. The four basic loops are applied through the successive straight line sections. It is possible to treat each section separately simply because all the solutions (routes) are constrained to pass through the angle tower site at the end of a section. Height alternatives are, of course, considered. It is obvious that the solution developed in this way is a tree from left to right and is rooted at the initial starting node. As in any other mathematical programming application it is important to be able to consider the conditions of infeasibility. As at a particular test site and tower height it may not be at all possible to obtain a feasible linkage, a suitable feasibility marker is set up to detect this condition. As the computation progresses (this node should be ignored) and this effict is achieved by associating an excessively large cost to the solution for this unconnected node. Fig. 3.5 illustrates how the solution tree is stored and may be used for the purpose of back tracking.

The computer program which was developed along these lines is given in Appendix 7. The program is written in EMA which was the most efficient autocode available on Atlas at the time of implementing the program.

c. Experience of using the program and discussions

The first rigorous test of the program was made on a twenty mile stretch of Dowlais Cowbridge 400kV super grid line over hilly terrain with considerable side slope. In this particular line route it was also required to site

L2 and L6 type towers according to the specifications of C.E.G.B. The solution was carefully examined by C.E.G.B. from both an engineering and economic stand point and they reported a saving of 7% on L2 type line and 5% on L6. The saving in cost per mile is thus of the order of £1,000, while the computing cost per run is only £15 per mile.

Subsequently a fifty mile stretch of 400kV Stella-Harker line was designed using this program, this time using only L6 type towers. These results have proven to be even more viable. A typical section of this line is shown in the enclosed profile drawings (supplementary materials). The sag curves and the towers shown in heavy lines represent the computer solution; those with broken lines represent the manual solution for this section as obtained by the Transmission Project Group of C.E.G.B. For the purpose of rigorous comparison the same initial and final angle towers should be used. However, these results are parts of the two complete solutions running through the whole length of the line. The sites and the types of towers obtained by each method are provided in the tower schedules (chart 3.2, 3.3). This particular section is unusually long, 21,844 feet = 4·13 miles, and led to the violation of a design specification. This specification was not mentioned when the program was being developed and hence does not appear in its implementation. It requires that no straight line section longer than three drum lengths of conductor, i.e. 3 x 4500= 13500 feet, should be strung without a tension tower. Hence, towards the end of this lengh at chainage 13200 and tower number 12, a D30E10 tower is used instead of the DE10 specified by the computer solution. Over this stretch of 4·13 miles the computer solution has reduced by two the number of towers used and has produced a saving of 36 units per mile, where the cost of the DM20 tower is taken as 100 units.

It is perhaps appropriate to consider in this section the optimality and the sensitivity of the solution in relation to the fineness of the grid used. The double span and uplift constraints (2·5), (2·6) imply that the

decision at s may not be completely justified unless suboptimals at feasible js are also examined. Another way of looking at this approximation is that the two state variables associated with these two constraints have not been considered, thus dimensionality is drastically reduced at the cost of exact optimality. Considered as a minimal routing problem, whenever a constraint of the above type rules out a trial linkage, it is necessary to investigate the next suboptimal (or better still an ordered set of suboptimals [1·26]) associated with the preceding node in order to arrive at the true optimal solution upto the current node. Although the need for, and the required order k, of such suboptimals may not be predicted, these could be efficiently handled by a list processing technique.

To study the sensitivity of the solution to variations in average spacing between test sites a number of line sections were run with both 100 ft and 50 ft spacing. In all the cases corresponding solutions had the same overall optimum cost, although some towers were shifted by 50 ft and occasionally the heights of the adjacent towers were interchanged in one solution relative to the other. One section of line was also run in the reverse direction, fixing the type and height of the angle tower at each end, and a solution identical with that for a forward run obtained.

For a line section of given length the computing time involved is a function of the grid sizes of the test sites and the clearance sites and the number of tower height alternatives available. An empirical equation relating the computing time may be expressed as

$$\text{number of computing instructions} = K.n^a.Q^b.r^c \qquad (3.1)$$

where K is some constant of proportionality,

n the average number of test sites in a given length, say 1000 ft,

Q the available tower height alternatives,

r the average number of clearance sites in 1000 ft.

Arguing from first principles it may be shown that

$$1 \leq a \leq 2,\ 1 \leq b \leq 2,\ 0 \leq c \leq 1. \tag{3.2}$$

In the present formulation, by defining every test site also as a clearance site it is implicitly assumed that

$$r \simeq p + n. \tag{3.3}$$

Computing time may be approximated more compactly by the equation,

$$\text{number of computing instructions} \simeq K\ n^{a_1}\ Q^b \tag{3.4}$$

where it is conjectured that $2 \leq a_1 \leq 3$. The relations (3.1), (3.2), (3.4) were not investigated in sufficient detail , and no reliable figures are available for these parameters. However, it may be mentioned that

1) From purely practical considerations all the tower height alternatives need to be considered, hence, the fineness of the grid in this direction is irrelevant.

2) For the various runs where the average grid size was halved from 100 to 50 feet, the number of computing instructions increased by a factor of just over four, which seems to agree with the bound $2 \leq a_1 \leq 3$.

d. Conclusions and suggestions for further work

It has been shown that the choice of the sites and heights of towers in the design of an overhead transmission line may be posed as a combinatorial (minimizing) problem and solved by applying the discrete deterministic multistage decision technique of dynamic programming.

The algorithm used does not necessarily guarantee to produce the global optimum solution; however, dimensionality is reduced at the cost of exact optimality. A worthwhile strategy that naturally suggests itself is to make use of the sensitivity of the solution to the fineness of the grid and to seek to improve the solution by re-applying the original algorithm to a perturbation of the first solution on a still finer grid. In this way the elegance of the dynamic programming search procedure may be preserved.

To be specific, consider the solution shown in Fig. 3.6, and assume that this solution is produced in the first place by the application of the original dynamic programming algorithm. A very fine mesh of test sites in the neighbourhood of the optimal tower sites may now be defined (Fig. 3.7) and the dynamic programming algorithm re-applied. The process may be iterated if an improvement in cost is obtained. On very long sections with completely level ground profile, the optimal solution, not considering the angle towers, will be given by siting towers of height H_j at equal intervals of $d_j(max)$, such that

$$c_j = \min_{1 \le k \le Q} \frac{c(H_k)}{d_k(max)} \qquad (3.5)$$

where j is the index for which the expression on the right hand side is the minimum and $d_k(max)$ is the maximum span that the towers of height H_k can have without violating ground clearance (Fig. 3.8). Having obtained the most economic solution in this way for the level ground profile, one might test whether the iteration and the perturbation scheme mentioned above converges to such global optimal solutions. It may be noted that in the hypothetical situation considered above, for the number of spans tending to infinity, the cost per unit length of the line given by c_j tends to the cost per unit length of the optimal line. No work has been done to implement such schemes.

While handling fairly substantial design exercises it was realized that once the central algorithm is satisfactorily implemented most of the difficulty occurs in processing and checking the input data. All these point towards the development of a comprehensive data processing scheme. One might need to update card files and hence magnetic tape files because of data errors, or if towers need to be resited when wayleaves could not be obtained. Finally one might wish to obtain the finished design suitably plotted on an incremental plotter. A flow chart illustrating the mechanics of a fairly comprehensive design system along these lines is shown in Fig. 3.9.

With a little imagination this scheme may be extended to an online computer-aided design exercise. A design engineer with the help of such a facility could make a worthwhile design analysis.

Character code and the input format for the individual cards in the card file.

* (asterisk) test-site as well as clearance site card.

\- (minus) clearance site only.

\+ (plus) end cards for the details of elevations at the angle tower site.

\> (greater than) last card of the section containing section details

((open round bracket) start of cards with 3-numerical values.

) (close round bracket) end of cards with 3-numerical values.

] (close square bracket) end of card file input.

5-numeric card layout.

↕	A1	A2	A3	A4	A5
Character marker	xxxx.xxx Elevation at L-40' or =1 if right offset =2 if left offset	xxxx.xxx Elevation at c/Line or amount of horizontal offset	xxxx.xxx Elevation at R-40' or elevation of the point	xxx.xxx (clearance) clearance type digit =0 if normal =1 if offset	xxxxx.xxx chainage

3-numeric card layout

↕	A2	A4	A5	
Character marker	xxxx.xxx Elevation at c/Line	xxx.xxx (clearance) clearance type digit	xxxxx.xxx	A1 = A2 = A3.

Fig. 5.1.

University of London
Atlas Computing Service

Author G. MITRA.
Job number LRR91PAI
Title Hypothetical Section Record.

Cards ✓ Tape 5-hole 7-hole

Date
Page of

Comments.

*	276·000	275·8	275·3	28·00	0·0	initial test site and clearance site
-	275·6	275·5	274·6	28·00	100·0	clearance site only
C						beginning of 3-numeric card
-	274·0	28·0	200·0			clearance site only
-	275·0	28·0	300·0			
*	276·0	28·0	400·0			test site and clearance site.
⊃						end of 3-numeric card
*	276·1	276·4	276·9 continued	28·00	500·0	test site and clearance site
+	246·9	1·0	246·8	0·0	1700·0	Data for angle tower base interpolation at the end of section. section length = 1750 ft.
+	247·1	1·0	247·2	0·0	1750·0	
+	247·3	1·0	247·4	0·0	0·0	
+	247·0	1·0	247·2	0·0	50·0	
⟩	0·0	1600·0	1750·0	201·0	620·0	end record of section's length, angle of turn etc.

Fig. 3.2.

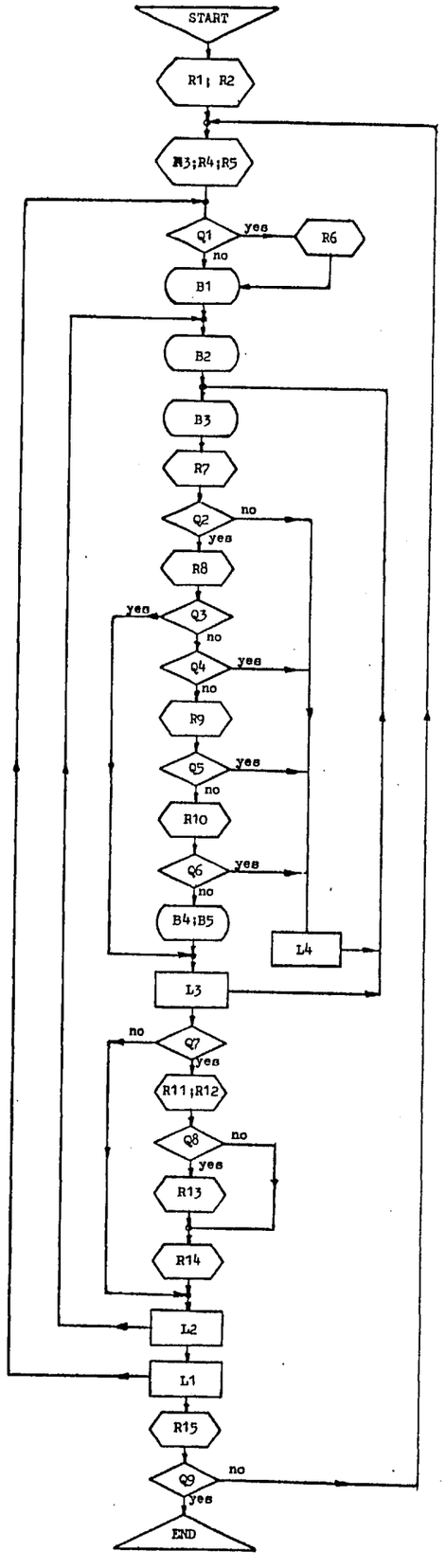


Fig. 3.3

Legend to Fig. 3.3.

Blocks

B1 Compute I and P the site indices over which the search for a previous tower is spread in the band of feasible connection.
B2 Initialize cost and solution markers before the search from a new node is started.
B3 Compute store index for the node to which connection is to be tried.
B4 Set the feasibility markers for the node solution and the site solution.
B5 Store the solution temporarily.

Decisions

Q1 Is there a change in maximum span in this section?
Q2 Is this connection worth trying, i.e. is there a feasible solution up to this node?
Q3 Is the present linkage of greater cost than a solution already obtained?
Q4 Is the double span constraint (2.5) violated?
Q5 Is the uplift constraint (2.6) violated?
Q6 Is the clearance marker reset by R10, i.e. is the clearance (2.16) violated?
Q7 Is there a feasible solution at this node?
Q8 Should there be a monitor output at this stage?
Q9 Is the end of line reached?

Loops

L1 Change test site indices 1(1)N.
L2 Change current tower height index 1(1)Q.
L3 Change previous tower site index I(1)P.
L4 Change previous tower height index 1(1)Q.

Subroutines

R1 Input the tower details.
R2 Generate the starting values controlled by input data.
R3 Bring down the section record from magnetic tape.
R4 Process the clearance-terrain data to transform them to elevation and side slope parameters.
R5 Initialize the starting parameters.
R6 Modify the span limit as required.
R7 Decode a solution with 'unpack' instructions.
R8 Compute the elements of the connection vector Y .
R9 Compute the sag curve at 22°F and check the uplift constraint.
R10 Compute the sag curve at 122°F and check the clearance constraint.
R11 Re-store the solution prior to final storage.
R12 Mark the solutions to ensure the ordering of cost with height.
R13 Stepwise monitor output of the solution.
R14 Store a solution with 'pack' instructions.
R15 Trace back linkage and output solution for each section.

Preceding site index j=0 (b_1)	Current site index s=N (b_2)	y_1	y_2	y_3	y_4	y_5
1 <u>true</u> $b_1 = 0$	<u>false</u> $b_2 = 1$	Evaluated by the relations given below		y_1	y_2 +string length	Evaluated by the relation given below
2 <u>false</u> $b_1 = 1$	<u>false</u> $b_2 = 1$			y_1 +string length	y_2 +string length	
3 <u>false</u> $b_1 = 1$	<u>true</u> $b_2 = 0$			y_1 +string length	y_2	
4 <u>true</u> $b_1 = 0$	<u>true</u> $b_2 = 0$			y_1	y_2	

details of evaluating type of tower (suspension or angle, and angle of turn) are not shown.

$y_1 = u_j + H_k - W_k g_j$ denotes the elevation of the conductor at the preceding tower site (test site= X_j)

$y_2 = u_j + H_t - W_t g_s$ denotes the elevation of the conductor at the current tower site (test site= X_s)

$y_3 = y_1 + (\text{string length}) * b_1$ denotes the elevation of the point of suspension of the insulator string at the preceding tower site.

$y_4 = y_2 + (\text{string length}) * b_2$ denotes the elevation of the point of suspension of the insulator string at the current tower site.

$y_5 = F_j(X_j, H_k) + C(H_t)$ denotes the cost of the trial linkage.

Chart 3.1

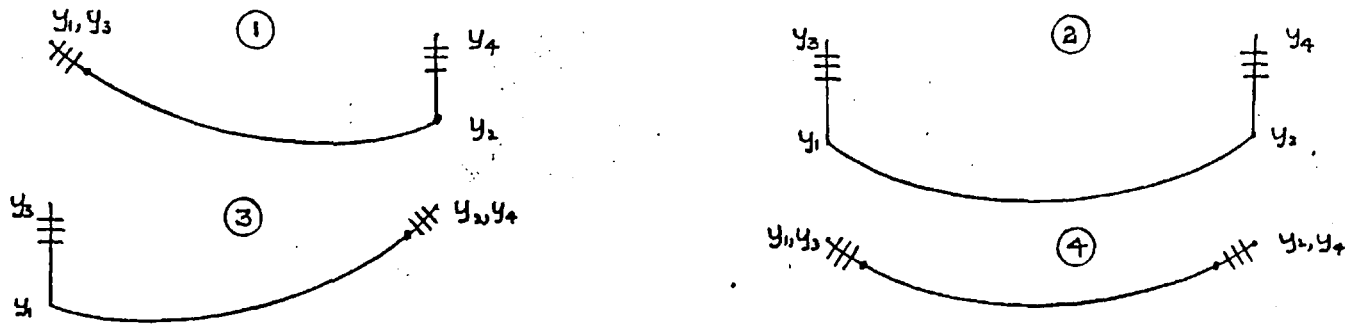
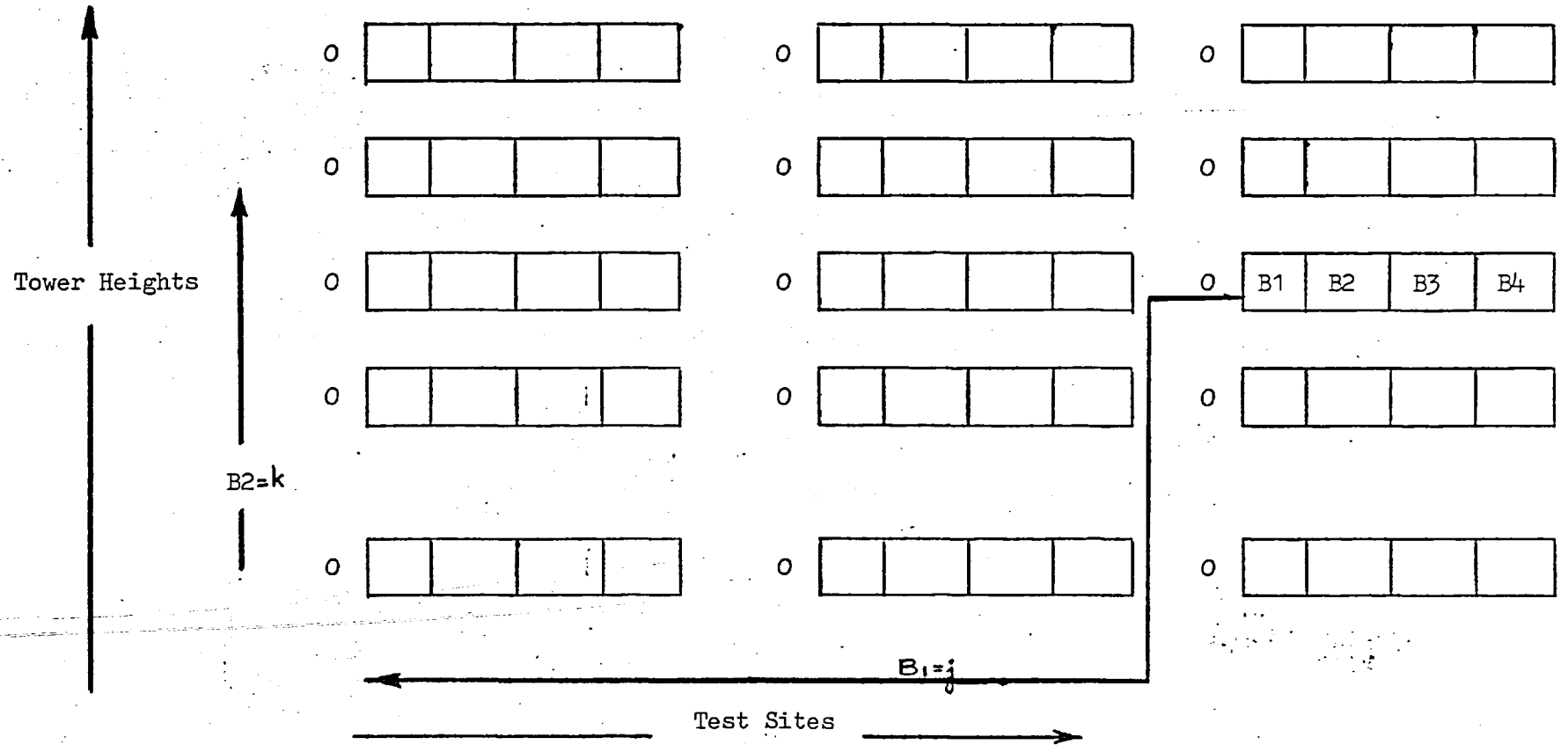


Fig. 3.4

Layout of the Store for the Storage of the Solution Tree



One typical pointer is shown to illustrate how the linked list is stored. From any feasible node one can trace back a solution to the origin. However, only the optimal solution is back tracked from the last angle tower.

Fig. 3.5

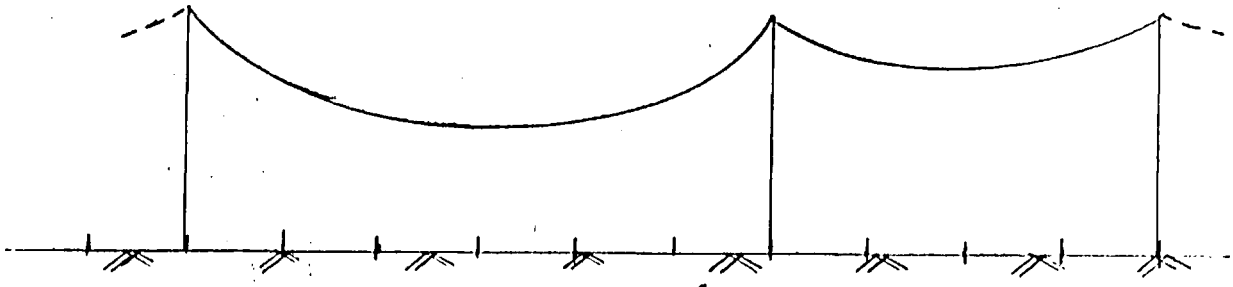


Fig.3.6.

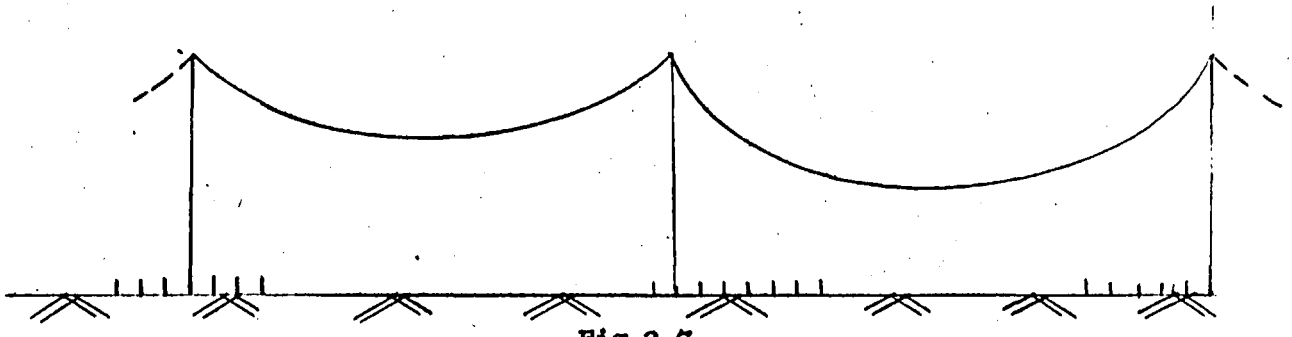


Fig.3.7.

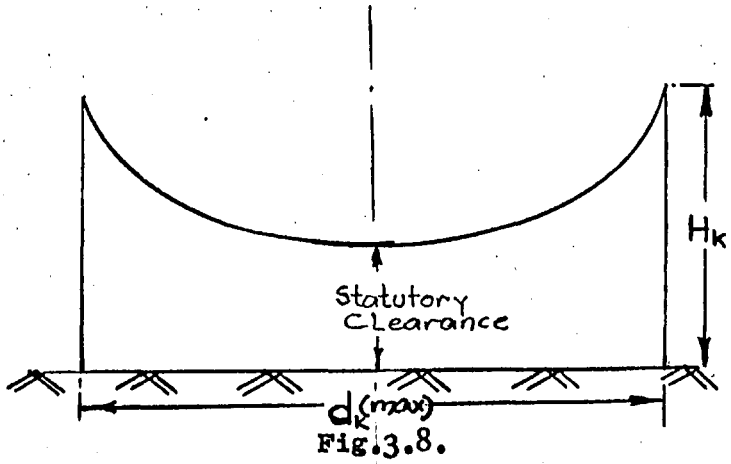


Fig.3.8.

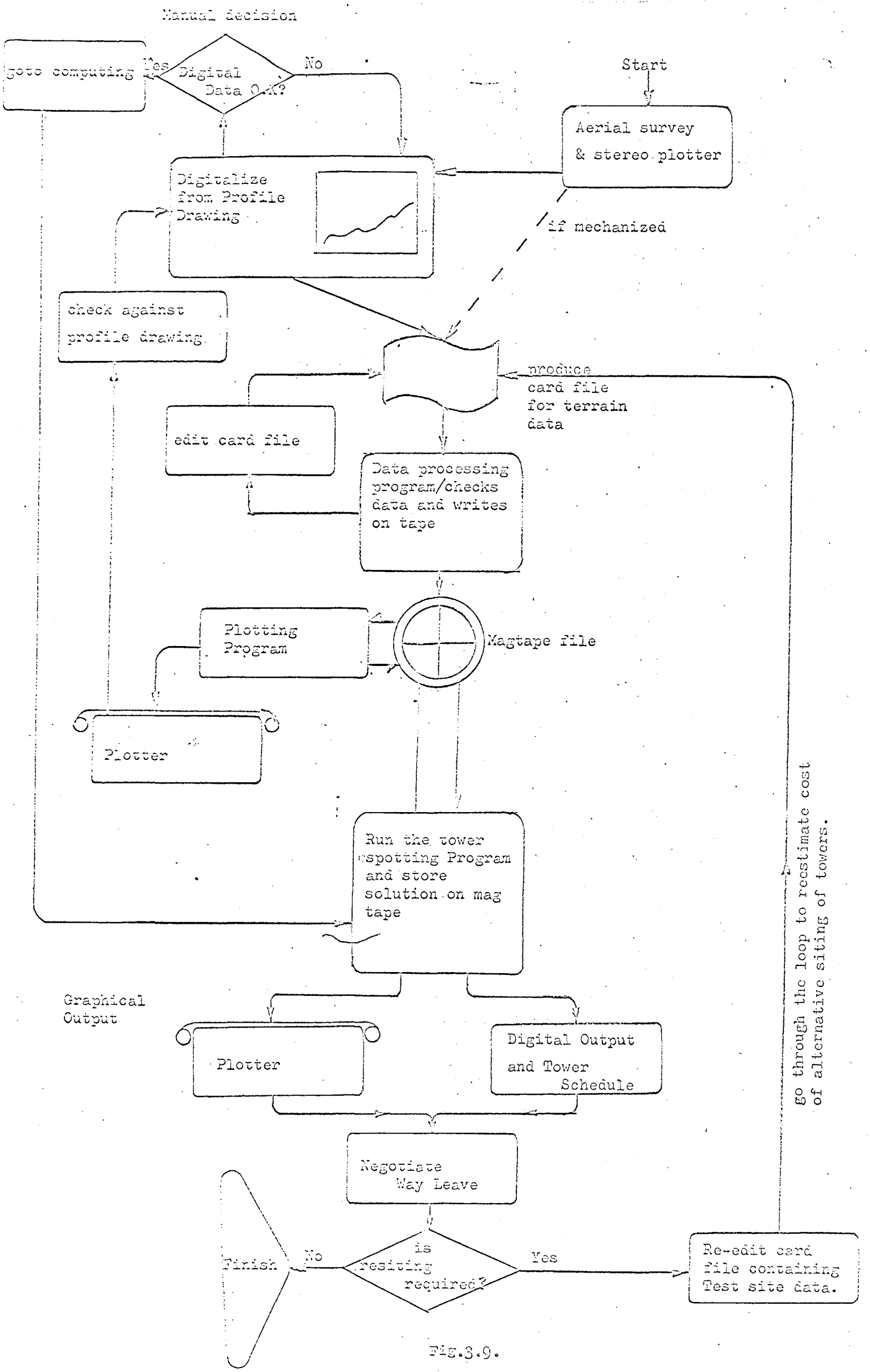

Manual decision
goto computing
Yes
Digital Data O.K?
No
Start
Aerial survey & stereo plotter
Digitalize from Profile Drawing
if mechanized
check against profile drawing
produce card file for terrain data
edit card file
Data processing program/checks data and writes on tape
Plotting Program
Magtape file
Plotter
Run the tower spotting Program and store solution on mag tape
Graphical Output
Plotter
Digital Output and Tower Schedule
Negotiate Way Leave
Finish
No
is resiting required?
Yes
Re-edit card file containing Test site data.
go through the loop to recstimate cost of alternative siting of towers.

Fig.3.9.

No.	Chainage Ft.	Tower Type	Base Level Ft.	Tower Height Ft.	Span Length Ft.	Cost	Remarks
1	0	D30STD	608·02	65·25	0	232	Strain (tension) insulator used
2	1200	DSTD	655·58	65·25	1200	111	
3	2000	DE10	622·53	75·25	800	116	
4	3500	DM10	653·73	55·25	1500	104	
5	4600	DE30	683·93	95·25	1100	129	
6	6000	DE30	701·36	95·25	1400	129	
7	7200	DSTD	676·92	65·25	1200	111	
8	8400	DE10	643·43	75·25	1200	116	
9	9720	DSTD	596·33	65·25	1320	111	
10	10770	DE30	516·87	95·25	1050	129	
11	12100	DM10	571·22	55·25	1330	104	
12	13200	D30E10	595·73	75·25	1100	242	Strain insulator used. Tower used as a tension tower (No.12)
13	14400	DE30	610·73	95·25	1200	129	
14	15800	DSTD	651·65	65·25	1400	111	
15	16900	DE10	702·24	75·25	1100	116	
16	18100	DE10	732·14	75·25	1200	116	
17	19400	DE20	774·63	85·25	1300	124	
18	20700	DE30	757·46	95·25	1300	129	
19	21844	D30M10	725·70	55·25	1144	218	Strain (tension) insulator used

Tower Schedule Obtained By Computer Method
Total Cost=2577
Chart 3.2

Tower Schedule Obtained By Manual Method - Total Cost = 2721 - Chart 3.3

No.	Chainage Ft.	Tower Type	Base Level Ft.	Tower Height Ft.	Span Length Ft.	Cost	Remarks
1	0	D3OM10	-	55·25	0	218	Strain (tension) insulator used
2	1150	DSTD	-	65·25	1150	111	
3	2110	DE10	-	75·25	960	116	
4	3480	DM20	-	45·25	1370	100	
5	4340	DSTD	-	65·25	860	111	
6	5100	DE10	-	75·25	760	116	
7	6260	DSTD	-	65·25	1160	111	
8	7260	DSTD	-	65·25	1000	111	
9	8390	DE10	-	75·25	1130	116	
10	9720	DSTD	-	65·25	1330	111	
11	10730	D2E20	-	85·25	1010	126	Non standard tower with extension
12	12080	DM20	-	45·25	1350	100	
13	12970	DSTD	-	65·25	890	111	Strain insulator used. Tower used as a tension tower (No 14)
14	13830	D3OSTD	-	65·25	860	232	
15	14610	DSTD	-	65·25	780	111	
16	15950	DE10	-	75·25	1340	116	
17	17080	DE10	-	75·25	1130	116	
18	18440	DE20	-	85·25	1360	124	
19	19630	DE10	-	75·25	1190	116	
20	20780	DE10	-	75·25	1150	116	Strain(tension) insulator used
21	21844	D3OSTD	-	65·25	1064	232	

PART 1

References

1.1 Hancock, H., Theary of Maxima and Minima, Ginn & Co. 1917, Dover Publications Inc. (1960).

1.2 Bliss, G.A., Calculus of Variations, Carus Mathematical Monographs 1925. Reprinted (1962).

1.3 Birkhoff, G., and Maclane, S.A., A Survey of Modern Algebra, New York, Macmillan. (1941).

1.4 Dantzig, G.B., Linear Programming and Extensions, Princeton University Press (1963).

1.5 Wolfe, P., Review of non-linear optimization, paper presented at the conference on Optimization, organised by I.M.A. 25 - 28 March 1968. Keele, England.

1.6 Symposium on Optimization, SIAM (1968) national meeting, January 11 - 14. An introduction to the general area of papers and discussions.

1.7 Arrow, K.J., Hurwicz, L., Uzawa, H., Studies in Linear and Non-linear Programming, Stanford University Press, (1958).

1.8 Hadley, G., Non-Linear and Dynamic Programming, Addison-Wesley Co. Inc., (1965).

1.9 Abadie, J., Non-Linear Programming, North Holland Publishing Co. - Amsterdam, (1967).

1.10 Dorn, W.S., Duality in Quadratic Programming, Quarterly of Applied Mathematics, 8 (1960), 155-162.

1.11 Wolfe, P., A Duality Theorem for Non-Linear Programming, Quarterly of Applied Mathematics, 19 (1961), 239-244.

1.12 Carrol, C.W., The Created Response Surface Technique for Optimizing Non-Linear Restrained Systems, Operations Research, 9 (1961), 2, 169-184.

1.13 Fiacco, A.V., and McCormick, G.P., The Sequential Unconstrained Minimization Technique for Non-Linear Programming, a Primal - Dual Method, Management Science, 10 (1964), 2, 360-366.

1.14 Saaty, T.L., and Bram, J., Non-Linear Mathematics, McGraw-Hill, (1964) 93-174.

1.15 Beale, E.M.L., Unconstrained Optimization, Lecture Notes Imperial College (1967-68).

1.16 Fletcher, R., and Powell, M.J.D., A rapidly Convergent Descent Method for Minimization, The Computer Journal, 6 (1963), 163-168.

1.17 Bellman, R., Dynamic Programming, Princeton University Press, Princeton New Jersey, (1957).

1.18 Bellman, R., Adaptive Control Processes: A Guided Tour, Princeton University Press, New Jersey (1961).

1.19 Bellman, R.E., and Dreyfus, S.E., Applied Dynamic Programming, Princeton University Press, Princeton, New Jersey (1962).

1.20 Held, M., and Karp, R.M., A Dynamic Programming Approach to Schedule Sequencing Problem, SIAM Journal, 10 (1962), 196-210.

1.21 Bowman, E.H., Assembly-Line Balancing by Linear Programming, Operations Research, 8 (1960), 385-389.

1.22 Tonge, F.M., Summary of A Heuristic Line Balancing Procedure, Computers and Thought, Ed, Feigenbaum, E.A., and Feldman, J., McGrawhill (1963) 168-190.

1.23 Doran, J., and Michie, D., Experiments With The Graphs Traverser Program, Experimental Programming Reports; No. 11. (1965) University of Edinburgh.

1.24 Ahlborg, F., and Palm, Y., Optimum Tower Spotting on Transmission Lines by means of Electronic Computer, Blue-White Series No. 33, Swedish State Power Board, (1962).

1.25 Bartelink, L.H., Spotting of Transmission Towers by Computer, Canadian Engineering Journal, July (1964), 25-32.

1.26 Bellman, R., and Kalaba, R., On k^{th} Best Policies, SIAM Journal 8, (1960), 582-586.

1.27 Converti, V., Hyland, E.J., and Tickle, D.F., Optimized Transmission Tower Spotting on Digital Computer, A.I.E.E., Power Apparatus and Systems, No. 59, April (1962), 55-63.

1.28 Popp, R.E., Dabekis, C.J., and Fullerton, F.M., Electronic Computer Program Permits Optimized Tower Spotting of Electric Transmission Towers, A.I.E.E. Power Apparatus and Systems, No. 66, June (1963), 360-365.

1.29 Ranyard, J.C., A Method to Minimize the Cost of Erecting the Towers in an Electric Power Transmission Line, Postgraduate Diploma Dissertation, University of Leeds, (1963).

1.30 Shulman, H., The Pole Spotting Problem, SIAM Review, 4 (1962), 12-15.

1.31 Hoare, P.J., and Morwood, C.H., Overhead Transmission Line Optimized Tower Positioning by Digital Computer, Journal of the Institute of Engineers, Australia, 36 (1964), 115-123.

1.32 Stagg, G.W., and Watson, M., Dynamic Programming - Transmission Line Design, I.E.E.E. International Convention Record (U.S.A.) 12 (1964), 3, 55-61.

1.33 Gortsko, A.B., Optimal Location of Factories, Translated from Mathematical Models and Optimal Planning, Nauka, Novosibirsk, (1966) paper presented at seminar on dynamic programming December (1966) Birmingham.

1.34 Dale, P.J., Realligning Underground Railway Tunnels, paper presented in the above seminar.

1.35 Jefferis, R.P., and Fegley, K.A., Application of Dynamic Programming to Routing Problems, I.E.E. Transactions on Systems Science and Cybernetics, 1, November (1965), 21-26.

1.36 Groboillot, J.L., et Deligny, J.L., Optimalisation des Projets Routier, paper presented at the 4th international conference on operational research (1966).

1.37 Howard, R.A., Dynamic Programming and Markov Processes, Technology Press, Cambridge, Massachusetts, (1960).

1.38 Central Electricity Generating Board, Transmission Project Group, Burymead House, Guildford. Written communication, March (1965).

1.39 Gracey, G.C., Overhead Electric Power Lines, Ernest Benn Ltd., London, (1963).

PART 2

Non-linear Mathematical Programs

Introduction

Two specific types of non-linear mathematical programming problems are discussed in this part of the thesis. The integer linear programming problem described in Chapter 4 is non-linear since the otherwise linear objective function and the constraint functions considered are only defined over an integral domain (the lattice points in the continuous linear space) and the first derivatives of these functions are consequently undefined. The cutting plane approach for the solution of these problems is first introduced, and an improved algorithm which makes use of another, 'Positive Diophantine', for the solution of a diophantine equation in non-negative variables is described. This approach appears to offer certain advantages over the existing algorithms of the cutting plane type. Finally, the theory has been extended to devise a direct method for the solution of pure integer programs; this method is similar to that outlined in [2.8].

In Chapter 5 the computer implementation of (a) convex quadratic programming algorithm is described. The method is due to Van-de-Panne and Whinston [2.15] and is very attractive in that the symmetry/skew symmetry property of a quadratic programming tableau is never destroyed. The parametric facility incorporated in the code is subsequently used to develop the tree search schemes considered in the Part 3 of the thesis. In Chapter 6 two applications of the quadratic programming code are considered. One of the applications, viz, the design of optimal vertical profiles of highways formulated as a quadratic programming problem, is claimed to be quite novel.

Chapter 4

An Improved Algorithm for the Solution of Integer Programs by the Solution of Associated Diophantine Equations.

Introduction

Formulation of certain classes of combinatorial problems and problems of other types, as integer or mixed integer linear programming problems is well known in the literature and adequately dealt with in text books [1.4], [1.8], [2.3]. One of the established methods for their solution viz, the cutting plane method first devised by Gomory [1958] and reproduced in [2.6], is also widely known. However, for certain problems, and this has no relation to their size, the method seems to require unusually large computing effort; this despite a formal proof for its convergence. In trying to visualize the situations in which such difficulties arise and if possible to counter these, the present algorithm was developed, and it may be claimed that it has been at least partly successful.

In Section a of this chapter the problem is stated, in Section b the solution to the continuous problem which is the first step towards solving the integer problem is considered. Section c discusses the relationships by which cuts are generated and how the proposed method produces improved cuts. Section d describes the algorithm 'Positive Diophantine', and Section e outlines the direct method for the solution of the integer problem from the continuous problem by the application of the above algorithm. Worked examples are provided in sections f and g ; Section h presents results and discussions. The investigation reported in this chapter was jointly carried out with Dr D.B.C. Richards, when he was a student at the Institute of Computer Science under the supervision of Mr K. Wolfenden.

a. Statement of the problem

Consider the problem of maximizing

$$x_0 = a_{00} + \sum_{j=1}^{n} a_{0j} (-x_j) \tag{4.1}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \leq a_{i0} \quad , \quad i = 1,2,\ldots,m \tag{4.2}$$

and $x_j \geq 0$, $x_j \equiv 0 \bmod(1)$ $\quad j = 0,1,2,\ldots,n,$

where for convenience it is assumed that all $a_{ij} \equiv 0 \bmod(1)$.

Introducing slack variables $x_i \equiv 0 \bmod(1)$, $x_i \geq 0$, $i = n+1, n+2, \ldots, n+m$, and adding a set of trivial equiations $x_j = -(-x_j)$, $j = 1,2,\ldots,n$ leads to the Tucker-Beale system

$$\begin{aligned} x_0 &= a_{00} + \sum_{j=1}^{n} a_{0j} (-x_j) \\ x_s &= 0 + \sum_{j=1}^{n} -\delta_{sj} (-x_j) \quad , \quad s = 1,2,\ldots n \\ x_i &= a_{i0} + \sum_{j=1}^{n} a_{ij} (-x_j) \quad , \quad i = n+1, n+2, \ldots n+m, \end{aligned} \tag{4.3}$$

where δ_{sj} is the Kronecker delta.

b. The continuous solution

To obtain the continuous optimum solution to the problem, the primal or dual simplex algorithm is applied, depending on whether the problem is dual or primal infeasible. The choice of the pivots by these rules are considered in Appendix 3 and the rules of the pivotal transformation are described in Appendix 5. The solution to the continuous problem is contained within the form

$$\left.\begin{aligned} Dx_0 &= \bar{a}_{00} + \sum_{p \notin B} \bar{a}_{0p} (-x_p) \\ Dx_i &= \bar{a}_{i0} + \sum_{p \notin B} \bar{a}_{ip} (-x_p) \quad , \quad i = 1,2,\ldots,n+m, \end{aligned}\right\} \tag{4.4}$$

where B denotes the set of indices of the vectors forming the current basis, viz, the basis of the optimal program $C(\bar{A})$. The coefficients D, the modulus of the determinant of the current basis, and $\bar{a}_{ij}$ the elemts of the matrix $\bar{A}$, are all integral. Setting the non-basic variables x_p to zero the continuous optimum solution is $x_i = \bar{a}_{io}/D$.

Let $I(\bar{A})$ denote the optimal integer program of the stated problem. If $\bar{a}_{io} \not\equiv 0 \bmod(D)$ for any $i = 0,1,2,\ldots m+n$, then $I(\bar{A}) \neq C(\bar{A})$.

c. Cutting plane steps

Let e be the index of any row of the system (4.4) such that $\bar{a}_{ep} \not\equiv 0 \bmod(D)$ for at least one $p \notin B$. For such a row define the fractional elements,

$$\left.\begin{aligned} f_p &= \frac{\bar{a}_{ep}}{D} - \left[\frac{\bar{a}_{ep}}{D}\right], \; p \notin B, \text{ so that } 0 \le f_p < 1, \\ \text{and} \quad f_o &= \frac{\bar{a}_{eo}}{D} - \left[\frac{\bar{a}_{eo}}{D}\right], \qquad 0 \le f_o < 1 \end{aligned}\right\}. \tag{4.5}$$

As in [2.4] this leads to the introduction of the reduced inequality,

$$f_o \le \sum_{p \notin B} f_p x_p \tag{4.6}$$

which must be satisfied by any feasible solution to the problem. However, if the fractional parts are expressed as ratios of integers with common denominator D, then under certain condition a stronger inequality can be constructed. From (4.5) one can obtain non-negative integers d_o, d_p such that

$$f_o = \frac{d_o}{D}, \; f_p = \frac{d_p}{D}, \; 0 \le d_o, d_p < D, \tag{4.7}$$

and substituting in (4.6)

$$d_o \le \sum_{p \notin B} d_p x_p. \tag{4.8}$$

Of course, this is just one of a series of parallel 'cuts' given by

$$d_o + r D \le \sum_{p \notin B} d_p x_p, \; r = 0, 1, 2, \ldots, \tag{4.9}$$

and such that the larger the value of r the deeper the cut into the convex region (4.2). It would appear desirable, therefore, to be able to determine the largest possible value of r such that the corresponding cut does not excclude any feasible lattice point. However, the best that can be done is to find the minimum value of r for which the diophantine equation

$$d_0 + r\,D = \sum_{P \notin B} d_p\, x_p \qquad (4.10)$$

has a non-negative integer solution in the variables x_p. Note that in the case of a primary cut (r=0) the diophantine equation always admits of solution in integer variables unrestricted in sign.

Formally, the problem now is to find minimum r and $x_p \equiv 0 \bmod(1)$, $x_p \geq 0$ such that (4.10) is satisfied. A dynamic programming solution has been proposed in [2.10] which, though conceptually elegant is hardly computationally efficient. An alternative approach is provided by the algorithm 'Positive Diophantine' (next section). The aim is to locate lattice points on the finite parallel planes (4.10) within the bounds $0 \leq x_p \leq \left[\dfrac{d_0 + r\,D}{d_p}\right]$. In most cases where cuts with $r > 0$ (secondary cuts) have been found to exist, the convergence to $I(\bar{A})$ has been more rapid than with primary cuts. Computationally worthwhile improvements have been observed in a number of test problems, Table 4.1.

d. <u>The algorithm 'Positive Diophantine'</u>

Consider the congruence

$$\sum_{i=1}^{n} \alpha_i\, x_i \equiv \alpha_0 \bmod(D) \qquad (4.11)$$

where $\alpha_i \equiv 0 \bmod(1)$ and $\alpha_i > 0$, $x_i \equiv 0 \bmod(1)$ and $x_i \geq 0$, i= 1,2,...,n. For convenience it may be assumed that the coefficients are ordered, so that $\alpha_1 \leq \alpha_2 \leq \alpha_3 \ldots \leq \alpha_n$. Let the diophantine equation

$$\sum_{i=1}^{n} \alpha_i\, x_i = \alpha_0 + r\,D = \phi_n \qquad (4.12)$$

define the problem P_n. Then set $x_n = \left[\frac{\phi_n}{\alpha_n}\right]$, so that the remaining x_i must satisfy

$$\sum_{i=1}^{n-1} \alpha_i x_i = \phi_n - \alpha_n \left[\frac{\phi_n}{\alpha_n}\right] = \phi_{n-1}, \tag{4.13}$$

defining the problem P_{n-1}. Repetition of the procedure leads to

$$\alpha_1 x_1 = \phi_2 - \alpha_2 \left[\frac{\phi_2}{\alpha_2}\right] = \phi_1, \tag{4.14}$$

defining the problem P_1; and further

$$\phi_1 - \alpha_1 \left[\frac{\phi_1}{\alpha_1}\right] = \phi_0. \tag{4.15}$$

may be defined. If at any stage of this process ϕ_k is zero, the congruence (4.11) is satisfied by the x_i already set for $i \geq k+1$, and $x_i = 0$ for $i \leq k$. If the entire sequence P_i, $i = n, n-1, \ldots 1$, is produced and $\phi_0 \neq 0$, then a new subsequence must be defined, starting at the last i for which $x_i > 0$. Suppose this was $i = k$, then reset $x_k = x_{k-1}$ and $\phi_{k-1} = \phi_{k-1} + \alpha_k$ and proceed with new subproblems P_i, $i = k-1, k-2, \ldots 1$. Whenever $\phi_i = 0$, the corresponding n vector is a lattice point on the hyperplane defined by (4.12). The recursive process can be continued until the search over the finite hyperplane is complete, that is, the components $x_n, x_{n-1}, \ldots, x_2$ are reduced to zero which indicates no solution exists.

e. <u>Direct determination of the integer solution $I(\bar{A})$ from the continuous solution $C(\bar{A})$.</u>

In the modified cutting plane method described in Section c one proceeds from $C(\bar{A})$ to $I(\bar{A})$ in the manner of [2.6] and at each cut generation stage applies the process outlined in the previous section. However, the additional pivotings involved may be avoided by taking advantage of the search technique of 'Positive Diophantine'.

Let

$$\bar{X} = \{ \bar{x}_i \mid \bar{x}_i = \bar{a}_{io}/D,\ i \in B;\ \bar{x}_i = 0,\ i \notin B \}$$

denote the n+m component solution vector of the continuous problem. Then corresponding to the optimum integer solution X^I (if it exists), there exists a non-negative integer vector of n components $x_p \equiv 0 \bmod(1)$, $p \notin B$, such that X^I can be expressed as,

$$X^I = \{x_i^I \mid x_i^I = \bar{a}_{io}/D + \sum_{p \notin B} (\bar{a}_{ip}/D)(-x_p),\ i \in B;$$
$$x_i^I = x_p,\ i = p \notin B\} \qquad (4.16)$$

where of necessity $x_i^I \equiv 0 \bmod(1)$ and $x_i^I \geq 0$, $i = 1,2,\ldots n+m$. The result presented in (4.16) has been stated and proved as a theorem by Gomory [2.8]. One pertinent interpretation of this result is that, to obtain the integer program directly it is necessary to determine the components of this all important n-vector of the variables x_p, $p \notin B$. The continuous optimum tableau is dual feasible and this ensures that all the coefficients $\bar{a}_{op}$, $p \notin B$ will be non-negative. While extracting the generalized cuts from the objective row (in the first two examples considered, Section g, these are parallel to the objective) it was realized that the objective row of $\bar{A}$ itself may be extracted and such a hyperplane parallel to the objective may be moved inside the convex by assigning different values of r until a feasible lattice point is located on it. If from the objective row of the matrix $\bar{A}$ one extracts the congruence

$$\sum_{p \notin B} \bar{a}_{op}\, x_p \equiv d_o \bmod(D)$$

and rewrites it in the form

$$d_o + r\,D = \sum_{p \notin B} \bar{a}_{op}\, x_p\ ,\ x_p \equiv 0 \bmod(1),\ x_p \geq 0,\ \ldots \qquad (4.17)$$

then it is to be noted, (4.17) poses the same problem as (4.8) viz, solution (in non-negative integers) of a diophantine equation with positive coefficients (for the time being it is assumed that $\bar{a}_{op} \neq 0$ for all $p \notin B$, the problem of dual degeneracy is dealt with in Section h).

For the purpose of clarity of exposition it is perhaps worthwhile to consider some features of this approach in detail. Suppose the tableau corresponding to the continuous optimum solution is reached and that this solution is not feasible (integrally) i.e., for some i, $\bar{a}_{io} \not\equiv 0 \bmod(D)$. Let the objective row be represented by

$$x_0 = \bar{a}_{oo}/D + \sum_{p \notin B} \left(\bar{a}_{op}/D\right)(-x_p). \tag{4.18}$$

Now define the hyperplane,

$$s_0 = -d_0 - r\,D - \sum_{p \notin B} \bar{a}_{op}(-x_p) \tag{4.19}$$

where $d_0 = \bar{a}_{oo} - \left[\frac{\bar{a}_{oo}}{D}\right] D$

The equation (4.19) represents a hyperplane parallel to the functional. If $\bar{a}_{op} < D$ for all $p \notin B$, and $r = 0$ then (4.19) defines Gomory's cutting plane extracted from the objective row. It is perhaps worth repeating that the non-negativity of $\bar{a}_{op}$ allows one to use the coefficients themselves (even when $\bar{a}_{op} \not< D$) and to define a cutting plane which the integer solution to the problem must satisfy. Now, $r = 1,2,\ldots$ in (4.19) define successive positions of the hyperplane such that the space between two adjacent positions is empty of feasible lattice points until for some $r = r^*$ (4.19) is a hyperplane containing a feasible lattice point. Note that once such an $r = r^*$ is obtained this provides the optimum solution. Clearly therefore the aim is to minimize r subject to (4.17) and at the same time satisfy,

$$\left.\begin{aligned} x_i &= \bar{a}_{io} + \sum_{p \notin B} \bar{a}_{ip}(-x_p) \equiv 0 \bmod(D), \\ \text{and} \quad x_i &\geq 0 \end{aligned}\right\} \text{for all } i \in B \tag{4.20}$$

As opposed to the usual approach of linear and integer linear programs one may check for this by generating the integer vector x_p, $p \notin B$ to satisfy (4.17) and then substitute explicitly in the tableau rows (4.20) to see if this is satisfied. Assuming $C(\bar{A})$ is not dual degenerate, that is,

objective row of $\bar{A}$ does not contain any zero element, an algorithm for obtaining $I(\bar{A})$ from $C(\bar{A})$ is outlined below

Step 1 Extract the hyperplane (4.19) parallel to the objective from the continuous optimum solution (4.4). Set r= 0.

Step 2 Apply the algorithm 'Positive Diophantine' to explore whether there exist $x_p \equiv 0 \bmod(1)$ and $x_p \geq 0$, $p \notin B$, which satisfy the equality of (4.17).

Step 3 Is the present hyperplane exhausted? If yes then r:= r+1; goto step 2.

Step 4 $x_p \geq 0$, $p \notin B$, are the components of an integer vector satisfying (4.17). Does this vector substituted in (4.20) satisfy the integrality and the non-negativity requirement of X^I? If yes then goto 6.

Step 5 Goto 2.

Step 6 Output the optimal integer solution and stop.

It may be noted that the algorithm can be simply modified to produce alternative optima if they exist. Feasibility or otherwise of the current program for each successive value of r can likewise be established with little extra effort.

f. Generalized cut;an example

Consider the problem (taken from [2.9]):

$$\begin{aligned} &\text{Minimize } x_2 \\ &\text{subject to } 33x_1 + 7x_2 \geq 715 \\ &\qquad -41x_1 + 14x_2 \geq 653 \\ &\qquad x_j \equiv 0 \bmod(1) \text{ and } x_j \geq 0,\ j = 1,2. \end{aligned} \tag{4.21}$$

Writing the tableau in the Tucker-Beale form and applying the dual simplex pivot rules the continuous optimum is obtained as follows:

		$-x_1$	$-x_3$
x_0	0	0	1
x_1	0	-1	0
x_2	0	0	-1
x_3	-715	(-33)	-7
x_4	-653	41	-14

D= 1

x_3, x_4 denote slacks;

Tableau 0

		$-x_3$	$-x_2$
x_0	0	0	33
x_1	715	-1	7
x_2	0	0	-33
x_3	0	-33	0
x_4	-50864	41	(-749)

D= 33

Tableau 1

		$-x_3$	$-x_4$
x_0	-50864	41	33
x_1	5439	-14	7
x_2	50864	-41	-33
x_3	0	-749	0
x_4	0	0	-749

D= 749

continuous optimum

Tableau 2

Using the objective row to generate the cut and by relation (4.9)

$$68 + 749\, r \leq 41x_3 + 33x_4 . \qquad (4.22)$$

Taking r= 0 for this and all the subsequent cutting planes (Gomory's method [2.6]), the optimum integer solution was obtained after 32 pivot steps. However, applying 'Positive Diophantine' to (4.22) as described in sections c and d, one obtains min(r)= 2 and the corresponding generalized cut is appended to Tableau 2 (of Tableau 2a).

		$-x_3$	$-x_4$
x_0	-50864	41	33
x_1	5439	-14	7
x_2	50864	-41	-33
x_3	0	-749	0
x_4	0	0	-749
s_1	-(68+2×749)	-41	(-33)

Tableau 2a

D= 749

Note: A random choice is made to break the tie for the selection of pivot column

		$-x_3$	$-s_1$
x_0	-2310	0	33
x_1	225	-1	7
x_2	2310	0	-33
x_3	0	-33	0
x_4	1566	41	-749
s_2	-(27+5×33)	-32	-7

Tableau 3

D= 33

Note: Second row is used for cut extraction. Applying 'Positive Diophantine' one obtains min(r)= 5 for which there exists a solution in positive integers for x_3 and s_1

		$-s_2$	$-s_1$
x_0	-2240	0	32
x_1	224	-1	7
x_2	2240	0	-32
x_3	192	-33	7
x_4	1280	41	735

Tableau 4

D= 32

Note: The optimum integer solution obtained in 4 iteration is

$$\min x_3 = \frac{2240}{32} = 70$$

$$x_1 = \frac{224}{32} = 7$$

$$x_3 = \frac{192}{32} = 6$$

$$x_4 = \frac{1280}{32} = 40$$

g. Direct method; examples

Consider the continuous optimum tableau of the previous problem (Tableau 2).

		$-x_3$	$-x_4$
x_0	-50864	41	33
x_1	5439	-14	7
x_2	50864	-41	-33
x_3	0	-749	0
x_4	0	0	-749

D= 749

Let the equation

$$68 + 749\, r = 41x_3 + 33x_4 \qquad (4.23)$$

be extracted from this tableau as explained in section e (note that this is the same as the cut extracted from the objective row in the first cutting plane step of the last section). Applying the algorithm 'Positive Diophantine' to the equating (4.23) one obtains min(r)= 2 and the corresponding solution

$$68 + 2(749) = 41(6) + 33(40), \text{ i.e. } x_3 = 6,\ x_4 = 40.$$

Substituting this in the continuous tableau as in (4.20)

$$749(x_0) = -50864 - 41(6) - 33(40) = -70(749) \equiv 0 \bmod(749)$$

$$749(x_1) = 5439 + 14(6) - 7(40) = 7(749) \equiv 0 \bmod(749) \geq 0$$

$$749(x_2) = 50864 + 41(6) + 33(40) = 70(749) \equiv 0 \bmod(749) \geq 0$$

$$749(x_3) = 0 + 749(6) + 0 = 6(749) \equiv 0 \bmod(749) \geq 0$$

$$749(x_4) = 0 + 0 + 749(40) = 40(749) \equiv 0 \bmod(749) \geq 0$$

the solution $I(\bar{A})$ is obtained directly from $C(\bar{A})$.

Consider next the application of this direct method to another problem from [2.9]:

Minimize x_3,

$$\left.\begin{aligned} \text{subject to } 5x_1 + 8x_2 - 7x_3 &\geq -89 \\ -6x_1 + 5x_2 + x_3 &\geq 11 \\ 3x_1 - 5x_2 + 2x_3 &\geq 29 \end{aligned}\right\} \quad x_j \equiv 0 \bmod(1),\ x_j \geq 0 \quad j = 1,2,3.$$

		$-x_1$	$-x_2$	$-x_3$	
x_0	0	0	0	1	
x_1	0	-1	0	0	D=1
x_2	0	0	-1	0	
x_3	0	0	0	-1	
x_4	89	-5	-8	7	
x_5	-11	6	(-5)	-1	
x_6	-29	-3	5	-2	

Tableau 0

		$-x_1$	$-x_6$	$-x_5$
x_0	0	0	0	5
x_1	0	-5	0	0
x_2	11	-6	-1	1
x_3	0	0	0	-5
x_4	533	-73	-8	43
x_5	0	0	-5	0
x_6	-200	12	5	(-15)

D= 5

Tableau 1

		$-x_1$	$-x_6$	$-x_5$
x_0	-200	15	5	5
x_1	0	-15	0	0
x_2	-7	-15	-2	1
x_3	200	-15	-5	-5
x_4	-121	(-90)	19	43
x_5	0	0	-15	0
x_6	0	0	0	-15

D= 15

Tableau 2

		$-x_4$	$-x_6$	$-x_5$
x_0	-1321	15	49	73
x_1	121	-15	-19	-43
x_2	79	-15	-31	-37
x_3	1321	-15	-49	-73
x_4	0	-90	0	0
x_5	0	0	-90	0
x_6	0	0	0	-90

D= 90

continuous optimum

Extracting the equation (4.17) from the objective row,

$$29 + 90r = 15x_4 + 49x_6 + 73x_8,$$

and applying the algorithm 'Positive Diophantine', Section d, one obtains the minimum r for which there exists a solution, namely min(r)= 3. Then

$$29 + 3(90) = 15(2) + 49(4) + 73(1),$$

i.e. $x_4 = 2,\ x_6 = 4,\ x_8 = 1.$

Substituting in the above tableau:

$$90(x_0) = -1321 - 15(2) - 49(4) - 73(1) = -18(90) \equiv 0 \bmod(90)$$

$$90(x_1) = 121 + 15(2) + 19(4) + 43(1) = 3(90) \equiv 0 \bmod(90),\ \geq 0$$

$$90(x_2) = 79 + 15(2) + 31(4) + 37(1) = 3(90) \equiv 0 \bmod(90),\ \geq 0$$

$$90(x_3) = 1321 + 15(2) + 49(4) + 73(1) = 18(90) \equiv 0 \bmod(90) \geq 0$$

$$90(x_4) = 0 + 90(2) + 0 + 0 = 2(90) \equiv 0 \bmod(90) \geq 0$$

$$90(x_5) = 0 + 0 + 90(4) + 0 = 4(90) \equiv 0 \bmod(90) \geq 0$$

$$90(x_8) = 0 + 0 + 0 + 90(1) = 1(90) \equiv 0 \bmod(90) \geq 0$$

which is the solution $I(\bar{A})$ to the integer problem.

Lastly, to bring out the relationship between the proposed method and dynamic programming, consider a cargo-loading problem taken from [1.19].

Maximize $72x_1 + 60x_2 + 40x_3 + 27x_4 + 20x_5 + 50x_6 + 85x_7 + 96x_8$

subject to, $20x_1 + 18x_2 + 14x_3 + 12x_4 + 10x_5 + 16x_6 + 22x_7 + 24x_8 \geq 100,$

$x_1, x_2, \ldots x_8 \geq 0$ and $x_1, x_2, \ldots x_8 \equiv 0 \bmod(1)$.

Using reduced tableau, that is, omitting the unit rows one obtains,

		$-x_1$	$-x_2$	$-x_3$	$-x_4$	$-x_5$	$-x_6$	$-x_7$	$-x_8$	
x_0	0	-72	-60	-40	-27	-20	-50	-85	-96	
x_8	100	20	18	14	12	10	16	22	(24)	D= 1

Tableau 0

		$-x_1$	$-x_2$	$-x_3$	$-x_4$	$-x_5$	$-x_6$	$-x_7$	$-x_8$	
x_0	9600	192	288	384	504	480	336	72	96	
x_8	100	20	18	14	12	10	16	22	1	D= 24

Applying 'Positive Diophantine', the problem of minimizing r subject to the non-negative integral solution of

$$0 + r(24) = 192x_1 + 288x_2 + 384x_3 + 504x_4 + 480x_5 + 336x_6 + 72x_7 + 96x_8$$

is now investigated. It is found that no such solution exists for r= 0,1,2, whereas for r= 3 the solution (0,0,0,0,0,0,1,0) is unique. Substituting in the constraint equation it follows

$$x_8 = 100 - 22(1) = 78 \not\equiv 0 \bmod(24).$$

Next, r= 4 has the solution (0,0,0,0,0,0,0,1) but again the integrality requirement of the constraint equation is not satisfied

$$x_8 = 100 - 1(1) = 99 \not\equiv 0 \bmod(24).$$

The cyclic process of incrementing r, determining lattice points and testing the constraint is continued until r= 16. The solutions of the diophantine equation are now (2,0,0,0,0,0,0,0), (1,0,0,0,0,0,0,2), (0,0,0,0,0,0,0,4). The first two solutions do not satisfy the integrality condition for x_8, but the last one does, $x_8 = 100 - 1(4) = 96 \equiv 0 \bmod(24)$. Hence, $x_8 = 4$, $x_1 = x_2 = \ldots$ $x_7 = 0$ is the optimum solution to the problem with $x_0 = 384$.

h. Results and discussions

An algol program was written for the cutting plane method which employed a primal/dual algorithm for the solution of linear programming problems. The dual algorithm being necessary at the cutting plane steps. To this the procedure 'Positive Diophantine' was added to generate the generalized cuts. The results reported in this section are, however, due to Dr D.B.C. Richards[2.28] who carried out extensive experiments on difficult sample problems, the author's contribution being the development of the algorithm 'Positive Diophantine' reported in Section d. The computer code developed for the direct method reported in Section e is given in Appendix.8

The results of solving 30 problems by a matched pair of programs are given in Table 4.1, overall execution times for the optimum cut method and Gomory's Method were 65·0 secs and 85·7 secs respectively on I.C.T. Atlas Computer.

Problem No.	Dimensions		No. of Iterations to Integer Solution (including the continuous stage)	
	Constraints	Variables	Gomory's Method	Optimum Cut Method
1	4	5	25	8
2	4	5	18	18
3	4	5	43	19
4	4	5	13	7
5	6	5	1431	397
6	6	5	80	80
7	4	5	>2004	557
8	4	5	89	81
9	6	6	558	469
10	4	5	8	8
11	7	7	9	9
12	7	7	25	25
13	3	4	17	19
14	15	15	41	29
15	2	2	37	35
16	3	3	30	7
17	2	2	13	4
18	3	3	15	6
19	3	3	55	33
20	2	2	200	200
21	2	2	305	89
22	2	2	298	86
23	8	8	46	48
24	2	2	10	6
25	3	2	5	5
26	5	2	8	8
27	3	3	4	4
28	2	5	4	4
29	5	2	12	12
30	3	3	9	5

Table 4.1

The experience of using both the optimum cut method and the related direct method is limited, however, the following observations may be made.

1. The D-number of the problems investigated may become too large for single length arithmetic even before the continuous optimum $C(\bar{A})$ is reached, thus inhibiting the application of either method. To combat this an adaptive cut generation technique [2.7] which attempts to restrict D might be applied.

2. In the direct method, if the continuous optimum solution is dual degenerate, then there will exist an infinity of solutions for the diophantine equation extracted from the objective row, namely

$$d_0 + r\,D = \bar{a}_{01}\,x_1 + \bar{a}_{02}\,x_2 + \ldots \bar{a}_{0n}\,x_n. \qquad (4.24)$$

If $\bar{a}_{0i} = 0$ for at least one i, $1 \leq i \leq n$, x_i can assume any non-negative integer value. This difficulty might be resolved by holding the already determined components of a solution to (4.24) at their current values and varying the parametric values of the indeterminate components (corresponding to $\bar{a}_{0i} = 0$) in a lexicographically ordered search on the successive equations.

Chapter 5

Convex Quadratic Programming By Satisfying Kuhn-Tucker Conditions and Using a Symmetric Tableau.

Introduction

In this chapter, the theory and an implementation of the symmetric formulation of the simplex method of quadratic programming, due to Van-de-Panne and Whinston [2.15] are described. In Section a the problem is defined, the application of the Kuhn-Tucker conditions and the setting up of the tableau are considered and some properties of the symmetric/skew symmetric tableau are explained. In Section b the theory underlying the method, the pivot choice rules and the convergence of the algorithm are discussed. In Section c a small example is worked out to illustrate the steps of the method and how it works. In Section d a simple extension of the technique is considered, whereby a parametric change in a constant term of the right hand side, or that of a linear cost term, can be studied. Section e outlines some relevant points regarding the computer implementation of the algorithm. In Section f a few tests for the positive definiteness of a quadratic form are considered, and Gauss's method of solving the normal equations is discussed.

a. Maximization of a quadratic form subject to linear inequalities.

Consider the problem

$$\text{Maximize } f(x) = \sum_{j=1}^{n} p_j x_j - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_i x_j \qquad (5.1)$$

subject to the constraints

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \; ; \quad i = 1,2,\ldots m \qquad (5.2)$$

$$\text{and } x = (x_1, x_2, \ldots x_n)' \geq 0 \qquad (5.3)$$

where c_{ij} are the elements of a symmetric matric C whose quadratic form is positive definite. In matrix notation this can be written as

$$\text{Maximize } f(x) = p'x - \tfrac{1}{2} x'Cx, \tag{5.4}$$

$$\text{subject to } Ax \leq b \tag{5.5}$$

$$\text{and } x \geq 0. \tag{5.6}$$

Introducing slack variables, $y_1 = x_{n+1}$, $y_2 = x_{n+2}$, $\ldots$ $y_n = x_{n+m}$, the Lagrange multipliers u_1, $u_2, \ldots u_n$, and $v_1 = u_{n+1}$, $v_2 = u_{n+2}, \ldots v_m = u_{n+m}$, the Kuhn-Tucker conditions for a saddle point optimum may be expressed as, cf (1.12) to (1.17)

$$p - Cx - A'v + u = 0 \tag{5.7}$$

$$Ax + y = b \tag{5.8}$$

$$v'(Ax - b) = 0 \tag{5.9}$$

$$x'u = 0 \tag{5.10}$$

$$u, v, x, y \geq 0 \tag{5.11}$$

Substituting (5.7) in (5.4) one can obtain the following relationship for the objective function,

$$uo = 2f(x) = 2p'x - x'Cx = p'x - u'x + v'Ax$$

$$= p'x + b'v - u'x - v'y. \tag{5.12}$$

When (5.9) and (5.10) hold, the above simplifies to

$$uo = 2f(x) = p'x + b'v. \tag{5.13}$$

Substituting for y and v and re-arranging, these can be written out in full as

$$2f(x) = uo = 0 + \sum_{j=1}^{n} p_j x_j + \sum_{i=1}^{m} b_i u_{n+i} \tag{5.14}$$

$$u_j = -p_j + \sum_{i=1}^{n} c_{ij} x_i + \sum_{i=1}^{m} a_{ij} u_{n+i} \quad ; \; j = 1,2,\ldots n \tag{5.15}$$

$$x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_j \quad ; \; i = 1,2,\ldots m$$

$$\sum_{k=1}^{m+n} u_k x_k = 0 \tag{5.16}$$

and x_k, $u_k \geq 0$, k= 1,2,...n+m. (5.17)

Taking the x and u variables on the right hand side of the equations (5.14), (5.15), (5.16) , with a negative sign, these can be presented(as in the Tucker form for linear programs) in the format of Tableau 5.1 known as the set up tableau. If the x_j, j= 1,2,...n and u_k, k= n+1, n+2,...n+m

	x_0	$-x_1$	$-x_2$	...	$-x_n$	$-u_{n+1}$	$-u_{n+2}$	...	$-u_{n+m}$
$2f(x)=u_0$	0	$-p_1$	$-p_2$	...	$-p_n$	$-b_1$	$-b_2$	...	$-b_m$
u_1	$-p_1$	$-c_{11}$	$-c_{21}$	...	$-c_{n1}$	$-a_{11}$	$-a_{21}$	...	$-a_{m1}$
u_2	$-p_2$	$-c_{21}$	$-c_{22}$	...	$-c_{n2}$	$-a_{12}$	$-a_{22}$	...	$-a_{m2}$
...	...	...	$[\alpha_{ij}$; i, j= 0,1,2,...n+m]						
u_n	$-p_n$	$-c_{n1}$	$-c_{n2}$	...	$-c_{nn}$	$-a_{1n}$	$-a_{2n}$	...	$-a_{mn}$
x_{n+1}	b_1	a_{11}	a_{12}	...	a_{1n}	0	0	...	0
x_{n+2}	b_2	a_{21}	a_{22}	...	a_{2n}	0	0	...	0
...	...	...	...	...	...	...	...	...	...
x_{n+m}	b_m	a_{m1}	a_{m2}	...	a_{mn}	0	0		0

Tableau 5.1

are set equal to zero, then one obtains the u_j, j= 1,2,...n, and x_k , k= n+1, n+2,...n+m of the trial solution given by the $-p_j$, j= 1,2,...n, and b_{k-n}, k= n+1,...n+m. Henceforth, x_j, j= 1,2,...n+m, will be called the primal variables and u_j the corresponding dual variables. For every primal problem of maximizing f(x) in x, there exists a dual problem of minimizing G(u,v) subject to the relevant constraints [Appendix 4]. If there exists a feasible solution to the primal, then the optimal solution to both the problems (*C* positive definite) must exist, and may be obtained by satisfying the above conditions applying a simplex algorithm. Prior to presenting the algorithm, the following terminology, the properties of the problem and its tableau representation are considered [2.15].

As indicated in Tableau 5.1, let α_{ij} denote any element of this tableau, defined for i,j= 0,1,2,...n+m. Further, let the following sets be defined:

X_r - the set of indices of the x variables in the basis,

X_c - the set of indices of the x variables not in the basis plus the index 0.

Then

1) for any j, $j \in X_r$, if $x_j = \alpha_{jo} < 0$, the tableau and its corresponding solution is said to be primal infeasible,

2) for any j, $j \in X_c$, if $u_j = \alpha_{jo} < 0$, the tableau is said to be dual infeasible,

3) for any x-variable in the basis, the corresponding u-variable must stay out of the basis (5.16), (5.17), and all the algorithms attempt to preserve this complimentary condition,

4) there exist symmetry and skew-symmetry properties [2.15] of the tableau, given by the relationships

$$\alpha_{ij} = \alpha_{ji} \quad , \quad i \in X_r \ \underline{\text{and}} \ j \in X_r$$

$$\alpha_{ij} = -\alpha_{ji} \quad , \quad i \in X_r \ \underline{\text{and}} \ j \in X_c$$

$$\alpha_{ij} = \alpha_{ji} \quad , \quad i \in X_c \ \underline{\text{and}} \ j \in X_c$$

$$\alpha_{ij} = -\alpha_{ji} \quad , \quad i \in X_c \ \underline{\text{and}} \ j \in X_r$$

If the complimentary condition of the primal and the dual variables is preserved, any pivotal transformation on the set-up, or any subsequent tableau, will not destroy this symmetry/skew symmetry property [2.15].

b. The theory underlying the method, the pivot choice rules, and the convergence of the algorithm.

It is convenient to assume that one starts with a primal feasible tableau, (property 1, last section) although such an assumption is not essential. Then, for the $u_j = \alpha_{jo} < 0, j \in X_c$, the corresponding non-basic x_j variable has a negative reduced cost coefficient, and the value of the objective function will be increased by moving in this direction. However, maximum increase is obtained,

a) if it is not possible to move in this direction any further without violating a constraint, or

b) if the gradient $\frac{\partial f}{\partial x_j} = \alpha_{jo} - \alpha_{jj} x_j$ [when all the other non basic x_s are

$= - u_j = 0,$ set equal to zero]

that is, the function assumes its maximum value in this direction.

Consideration of the possibility (b) allows one to have trial solutions at the intersection of two planes, or on the face of a plane, or even right inside the convex, where all the x-variables can become basic; that is, the unconstrained maximum of f(x) satisfies all the constraints. Under certain conditions, these u variables are equivalent to free-variables in Beale's method and in his approach, these are introduced if and when necessary [2.5].

When a trial solution on the face of a constraint $x_j = 0$, is such that it is a tangent point and not a vertex of this face, the following situation might arise. In trying to make this variable basic (its reduced cost having been found to be negative) it is observed that, of the available directions of movement, the objective function can improve only by moving along the inward pointing ray obtained from the gradient direction (direction B-C in the example). However, if one tries to move

fully in such a direction, some constraint, say x_s, may be violated, $x_s < 0$, and such a variable is called a blocking variable. For the purpose of exposition, the following terminology is defined.

Standard Tableau:

A tableau in which the complimentary condition of the primal and dual variables (5.16) is satisfied, and displays a symmetric/skew symmetric property, is called a standard tableau.

Blocked Tableau:

This refers to a standard tableau in which the incoming variable cannot be introduced at the desired level because of a blocking variable (identified by a ratio test) of the type just mentioned.

Unblocked Tableau:

A tableau in which the above situation does not arise is called an unblocked tableau.

Incoming Variable:

A variable that one is trying to bring into the basis is called an incoming variable.

Outgoing Variable:

A variable that is going to be made non-basic by the next pivot operation is the outgoing variable.

Single Pivot:

When a u variable is exchanged for its corresponding x variable, the corresponding transformation is a single pivot operation, cf (Appendix 5).

Double Pivot:

When two variables x_s and u_t enter the basis, and the corresponding u_s and x_t leave the basis, such a transformation is called double pivot, (Appendix 5).

The rules for choosing the pivot elements, and when to apply the single and double pivot are as follows ([2.15]).

Selection of the incoming variable:

In an unblocked tableau, select the primal variable corresponding to the most negative dual variable. In a blocked tableau the incoming variable is the same as that in the previous tableau.

Selection of the outgoing variable:

Select the outgoing basic variable by a ratio test similar to that employed in the simplex method of linear programming. However, to perform this ratio test, in addition to the basic primal variables, the dual of the incoming primal variable must be considered. in other words, find an index $\emptyset$ such that,

$$\theta = \min \left(\frac{\alpha_{k,o}}{\alpha_{k,in}}, \frac{\alpha_{in,o}}{\alpha_{in,in}} \mid \alpha_{k,o} \geq 0 \text{ and } \alpha_{k,in} > 0,\ k \in X_r;\ \alpha_{in,o} < 0,\ \alpha_{in,in} < 0 \right)$$

$$= \frac{\alpha_{\emptyset,o}}{\alpha_{\emptyset,in}}, \text{ where 'in' denotes the index of the incoming variable.} \qquad (5.18)$$

Rules for single or double pivot operation:

If the outgoing variable is the dual of the incoming variable (that is $\emptyset$= in), then perform single pivot on $\alpha_{in,in}$; the next tableau is then unblocked. If the out-

going variable is a primal one and if $\alpha_{\phi,\phi}$ is non zero, perform a single pivot on $\alpha_{\phi,\phi}$; the next tableau is then blocked. If this element is zero, select the incoming variable, but perform a double pivot operation whereby x_{in} and u_{ϕ} are exchanged for u_{in} and x_{ϕ}. The next tableau is then unblocked.

This is a statement of the algorithm in the primal form; the dual algorithm is the same as above if the x_s are replaced by the u_s and primal by dual. [2.15]

In an earlier paper [2.14] Van-de-Panne and Whinston have developed a variant of Dantzig's algorithm [2.4] for convex quadratic programming, and have shown that the method converges in finite number of steps. The current method is conceptually similar and its convergence may be proved arguing along the following lines [2.15]. The solution of two successive unblocked tableaus must have increasing values of the objective function [2.15]. For the intermediate blocked tableaus, if the incoming variable is taken at the value at which blocking occurs, then again the non-decreasing property of the objective function and primal feasibility is preserved (cf example Section c). The standard tableaus (blocked or unblocked) do not repeat and their number is finite, as the number of the combinations of the basic and non-basic variables is finite; hence, the steps of the iteration must be finite.

c. A worked example

Consider the problem(taken from Beale [2.5] and modified)

$$\text{Maximize } f(x) = 6x_1 + 0 - \tfrac{1}{2}(4x_1^2 - 4x_1x_2 + 4x_2^2) \qquad (5.19)$$

subject to,

$$\left.\begin{aligned} x_1 + x_2 &\le 2 \\ x_1 &\le \tfrac{7}{4} \\ x_2 &\le 1 \\ x_1 \,,\, x_2 &\le 0 \end{aligned}\right\} \qquad (5.20)$$

The problem is set up in the format of tableau 5.2, and the algorihhm outlined in the last section is applied. Performing the ratio test, one obtains, $(u_1 = -6 \;,\; in = 1)$,

$\theta = \min\{ \frac{2}{1} , \frac{7}{4}/_1 , \frac{-6}{-4} \} = \frac{6}{4}$, hence, 'in'= $\emptyset$, the primal and dual variables u_1, x_1 are therefore exchanged. In tableau 5.3 $u_2 = -3$; therefore, x_3

	x_0	$-x_1$	$-x_2$	$-u_3$	$-u_4$	$-u_5$	
$2f(x) =$ u_0	0						
u_1	-6	(-4)					Solution A
u_2	0	2	-4				Unblocked Tableau
x_3	2	1	1	0			
x_4	$\frac{7}{4}$	1	0	0	0		
x_5	1	0	1	0	0	0	

Tableau 5.2

	x_0	$-u_1$	$-x_2$	$-u_3$	$-u_4$	$-u_5$	
u_0	9·00						
x_1	1·50	-0·25					Solution B
u_2	-3·00	0·50	-3·00				Unblocked Tableau
x_3	0·50	0·25	1·50	-0·25			
x_4	0·25	0·25	0·50	-0·25	-0·25		
x_5	1·00	0·00	1·00	0·00	0·00	0·00	

Tableau 5.3

needs to be brought into the basis, however, ratio test shows x_3 that blocks the increase of x_3 to $x_3 = 1$, which maximizes $f(x)$ in this direction by making $u_3 = 0$. One therefore pivots on $\alpha_{33} = -0{\cdot}25$ to make this variable zero (non basic); however, the resulting tableau 5.4 is a blocked tableau in which the incoming variable x_3 has to be taken at its blocking level, namely $x_3 = \frac{1}{2}/\frac{3}{2} = \frac{1}{3}$. This makes the solution feasible and the function value increases to $2f(x) = 10\frac{2}{3}$. On the blocked tableau 5.4, solution C, one still tries to bring x_3, the previous incoming variable, into the basis; by ratio test one finds that it is possible to exchange x_3 for u_3, hence, one pivots on $\alpha_{23} = -12$ and obtains the tableau 5.5. This tableau is unblocked, satisfies the Kuhn-Tucker condition of optimality, and provides the optimal solution given by the zero column, if all the non basic u and x variables are set equal to zero.

	x_0	$-u_1$	$-x_2$	$-x_3$	$-u_4$	$-u_5$
u_0	8·00		[to be taken at $x_2 = \frac{1}{3}$]			
x_1	2·00	0·00		Solution $\mathbf{c}(\mathbf{c}')$		
u_2	-6·00	-1·00	-12·00	Blocked Tableau		
u_3	-2·00	-1·00	-6·00	-4·00		
x_4	-0·25	0·00	-1·00	-1·00	0·00	
x_5	1·00	0·00	1·00	0·00	0·00	0·00

Tableau 5.4

	x_0	$-u_1$	$-u_2$	$-x_3$	$-u_4$	$-u_5$
u_0	11·000					
x_1	1·500	-0·083			Solution D	
x_2	0·500	0·083	-0·083		Unblocked Tableau [Optimal Tableau]	
u_3	1·000	-0·500	-0·500	-1·000		
x_4	0·250	0·083	-0·083	-0·500	-0·083	
x_5	0·500	-0·083	0·083	-0·500	0·083	-0·083

Tableau 5.5

The trial solutions A,B,C,D are shown in the diagram Fig. 5.1. Setting the incoming variable to a critical level in a blocked tableau, may also be interpreted as equivalent to introducing a cutting plane. Such an approach has important connotations; thus instead of depending upon the Kuhn-Tucker condition of optimality, if one uses a more direct optimality criterion, the method may be modified to produce a local optimum as in Beale's method, when the quadratic form is not positive (semi) definite.

d. Parametric change in the linear cost terms or in the constant terms of the constraints

In the most general form, the parametric quadratic programming problem may be stated as, investigate the maximization of

$$f(x) = (p + \lambda p^*)\, x' - \tfrac{1}{2} x' C x \qquad (5.21)$$

subject to

$$A x \leq b + \mu b^* \qquad (5.22)$$

$$x \geq 0$$

for the appropriate ranges of variation of the scalars λ and μ, given the vectors p, p*, b, b*, and the matrices C,A.

In this section, however, a much simpler but none the less non-trivial problem of changing one of the elements of p or b, say p_j or b_i at any one time is considered. The ability to handle such parameter changes has been used in obtaining minimum variance solutions for different expected profits and therefore to produce E-V diagrams(due to Markowitz [2.17] or more recently Paine [2.18]). One such application is considered in Chapter 6; the work reported in chapters 7 and 9 also depends critically on this parametric facility.

Suppose by the application of the algorithm mentioned in Section b, one has obtained the optimal solution to the original problem (5.1), (5.2), (5.3), and let this be contained in the currently available tableau of elements α_{ij}. If the linear cost coefficient p_s is altered to Π_s then the effect of this change will be as follows:

a) If u_s is in the basis, then the new values of the primal and dual solution vectors as given by the zero column of the tableau will be,

$$\begin{aligned} \alpha'_{jo} &= \alpha_{jo} \quad , \quad j \in X_r \\ \alpha'_{jo} &= \alpha_{jo} \quad , \quad j \in X_c \ \underline{\text{and not}}\ s \\ \alpha'_{so} &= \alpha_{so} + p_s - \Pi_s . \end{aligned} \tag{5.23}$$

In short, only the value of u_s will be altered. Setting this to zero one obtains,

$$\Pi_s^{uc} = \alpha_{so} + p_s ,$$

the upper critical value, after which the solution becomes dual infeasible, that is, non optimal. Obviously, there can be no lower critical limit for Π_s.

b) If u_s is not in the basis, then the new value of the zero column is given by the relationship,

$$\alpha'_{io} = \alpha_{io} - (\Pi_s - p_s)\,\alpha_{is} \quad , \quad i = 1,2,\ldots n+m. \tag{5.24}$$

The function value will be given by the relationship,

$$2f(x) = u_o' = \alpha_{oo} + 2\,(\Pi_s - p_s)\,\alpha_{so} - (\Pi_s - p_s)^2 \alpha_{ss} \tag{5.25}$$

Thus for $\alpha_{ss} = 0$, one obtains the linear programming relationship of the change in twice the objective function. The upper and lower critical values of Π_s are given by the relationships,

$$\Pi_s^{uc} = p_s + \min_{\alpha_{is}>0} \left(\frac{\alpha_{io}}{\alpha_{is}} \right) \tag{5.26}$$

and

$$\Pi_s^{Lc} = p_s - \min_{\alpha_{is}<0} \left| \frac{\alpha_{io}}{\alpha_{is}} \right| \tag{5.27}$$

If the constant terms b_q are replaced by β_q one obtains similar relationships with the signs changed, and since this is coded in the program in the procedure 'parametrize' (cf next section and Appendix 9) these equations are not repeated here. From the point of view of computation, these parameter changes imply that the zero column of the tableau must be updated, and the critical values calculated, and finally the basic algorithm (which is capable of dealing with primal and dual infeasibilities) applied to find the optimal solution to the modified problem.

For the problem solved in the last section, the effects of changing $b_1 = 1\frac{3}{4}$ to β_1 is now investigated. As a consequence of this change, $\alpha_{10} = 0{\cdot}25$ is altered to $\alpha'_{10} = 0{\cdot}25 - 1{\cdot}75 + 1{\cdot}00 = -0{\cdot}5$. The tableau 5.5 is updated to tableau 5.6, and the dual algorithm applied to obtain the optimal solution in two iterations shown in tableaus 5.7 and 5.8. The change in the linearly constrained region and the new solution are shown in the Fig. 5.2.

	x_0	$-u_1$	$-u_2$	$-x_3$	$-u_4$	$-u_5$
u_0	11·000					
x_1	1·500	-0·083				
x_2	0·500	0·083	-0·083			
u_3	1·000	-0·500	-0·500	-1·000		
x_4	-0·500	0·083	-0·083	-0·500	-0·083	
x_5	0·500	-0·083	0·083	-0·500	0·083	-0·083

Tableau 5.6

	x_0	$-u_1$	$-u_2$	$-u_3$	$-u_4$	$-u_5$
u_0	12·000					
x_1	2·000	-0·333				
x_2	1·000	-0·167	-0·333			
x_3	-1·000	0·500	0·500	-0·000		
x_4	-1·000	0·333	0·167	-0·500	-0·333	
x_5	0·000	0·167	0·333	-0·500	-0·167	-0·333

Tableau 5.7

	x_0	$-u_1$	$-u_2$	$-u_3$	$-x_4$	$-u_5$
u_0	9·000					
x_1	1·000	0·000				
x_2	0·500	0·000	-0·250			
x_3	0·500	0·000	0·250	-0·250		
u_4	3·000	-1·000	-0·500	1·500	-3·000	
x_5	0·500	0·000	0·250	-0·250	-0·500	-0·250

Tableau 5.8

Starting from the original problem and its optimal tableau 5.5, consider now the effect of changing $p_1 = 6$ to $\Pi_1 = 3$. Applying the equations (5.24) and (5.25), the new updated tableau 5.9 is obtained, representing the point Δ in the Fig. 5.3. The tableau is dual infeasible and hence the primal algorithm is applied to produce the dual feasible solution tableau 5.8 . The critical values of Π_1 are

$$\Pi_1^{uc} = 6 + {}^{\frac{1}{4}}/_{\frac{1}{12}} = 9$$

$$\Pi_1^{lc} = 6 - {}^{1}/_{\frac{1}{2}} = 4.$$

The optimal solution, F, Fig. 5.3 is also the unconstrained maximum of f(x).

A pertinent feature to consider is that, by making parametric change to the linear cost terms (constant terms), one only alters the location of the unconstrained optimum of the primal (dual) problem. The eigensystems of the primal and the dual problems contained in the quadratic form $\mathcal{E}$ remain unaffected by the process.

e. Computer implementation of the algorithm

Ideally, any simplex type algorithm that works on the principle of pivotal transformations should be implemented by a product-form-of-the-inverse matrix technique [2.5]. However, the interest of the present

	x_0	$-u_1$	$-u_2$	$-x_3$	$-u_4$	$-u_5$
u_0	2·750					
x_1	1·250	-0·083				
x_2	0·750	0·083	-0·083			
u_3	-0·500	-0·500	-0·500	-1·000		
x_4	0·500	0·083	-0·083	-0·500	-0·083	
x_5	0·250	-0·083	0·083	-0·500	0·083	-0·083

Tableau 5.9

	x_0	$-u_1$	$-u_2$	$-u_3$	$-u_4$	$-u_5$
u_0	3·000					
x_1	1·000	-0·333				
x_2	0·500	-0·167	-0·333			
x_3	0·500	0·500	0·500	-1·000		
x_4	0·750	0·333	0·167	-0·500	-0·333	
x_5	0·500	0·167	0·333	-0·500	-0·167	-0·333

Tableau 5.10

study has been mainly in extending the basic quadratic programming algorithm to implement other facilities such as branch and bound search, chapters 7 and 9. The incorporation of explicit pivotal transformations made the primal and the dual algorithm very easy to program. The 'symmetric quadratic program' written in Algol is given in Appendix 9 . The main features of the code are considered below.

Representation of the symmetric matrix:

The one dimensional array, a[0: (n+m+1) × (n+m+2)/2], is used to represent the symmetric matrix of the tableau. This array is given a second dimension by having an integer array of pointers Pt [0: m+n], which is set up by making Pt[0]:= 0; followed by the statement,

```
for i:= 1 step 1 until n+m do Pt[i]:= Pt[i-1] + i;
```

the scheme is shown in fig. 5.4 where a[Pt[i]+j] $i \geq j$, represents the elements a_{ij} of the symmetric lower triangular matrix.

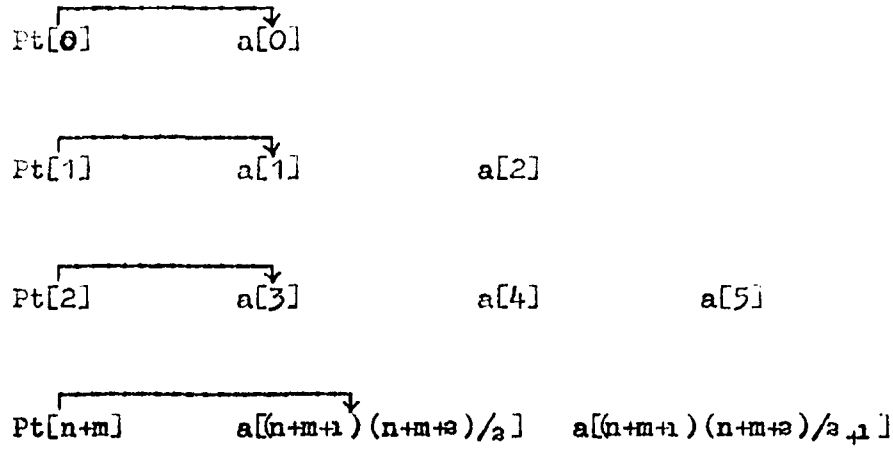


Fig. 5.4

The main identifiers are,

Tst - type boolean,and has the value true if the tableau is unblocked; it has the value false otherwise,

primal - type boolean, and has the value true if the tableau is primal feasible; it has the value false otherwise,

dual - type boolean,and has the value true if the tableau is dual feasible, it has the value false otherwise,

Phno - type integer. It has the value 1, if artificial variables are in the basis. It is set to value greater than 1 otherwise,

incoming - type integer, index of the incoming variable,

outgoing - type integer, index of the outgoing variable,

vcol[0:n+m] integer array, containing the character markers of the non-basic variables,

vrow[0:n+m] integer array, containing the character markers of the basic variables,

The main procedures are,

monitor - monitors intermediate or final tableaus if and when required,

pivot - performs single pivot operations by the rules given in Appendix 5, on a diagonal non zero entry of the symmetric tableau,

duopivot - performs a double pivot operation whereby variables given by the indices 'incoming' and 'outgoing' can be exchanged for their corresponding basic variables, cf Appendix 5. The tableau after being transformed by this procedure continues to possess symmetric/skew symmetric property.

Since row (basic) and column (non-basic) character markers are used, and the ordering of the variable - indices preserved, no unit rows or columns are required to keep track of the variables.

Finally the procedure,

parametrize - evaluates and monitors the critical values of the parameter changed, and updates the zero column of this tableau, and then transfers control to the main body of the program, where the primal and dual algorithms are applied in sequence.

The strategy of applying the algorithm, to handle, equality constraints, dual and primal infeasibilities in sequence, is shown in Fig. 5.5. In phase 1, when Phno= 1, the artificial variables are in the basis and the corresponding rows are scanned for non zero elements in the x-columns. A double pivot operation exchanging the x and u variable of the columns and

the rows is performed to make the artificial variables non basic, and subsequently these rows and columns are dropped. At the end of this stage Phno is set greater than 1; the tableau might be primal and dual infeasible.

It is well known that a primal algorithm cannot be applied to resolve dual infeasibility if the tableau is not primal feasible. To minimize the sum of the primal infeasibilities requires enlarging the problem by at least one more row and column. One approach could be to pivot on the diagonal elements of this quadratic form to arrive at the unconstrained optimum where all the dual variables are made non basic, hence, the tableau is dual feasible. Such methods may be important for theoretical finesse, but turn out to be not so important from a practical consideration. A simple modification of the primal pivot choice rule was made, whereby primal infeasibilities (negative primal variables) for the purpose of ratio test are neglected which produced dual feasibility in all the problems tried. This however, requires ignoring 'unbounded' exit for the linear programming (semi definite) problems. Such an approach is easy to justify, since the equivalent problem that one is investigating is primal feasible. At the end of this step the tableau is dual feasible; the dual algorithm is then applied to produce primal feasibility and the optimal solutions. The parametric procedure may be applied on the tableau if and when necessary. Fig. 5.3 illustrates how the state of the tableau is irreversibly transformed by the application of the primal and the dual algorithm which produce these sequential transformations. The facility of resolving primal infeasibility last by the application of the dual algorithm has been made use of in developing the branch and bound algorithm reported in Chapter 7.

The input format, the method of use of the parametric facility, and the interpretation of the output data are described in the document T.N.4,

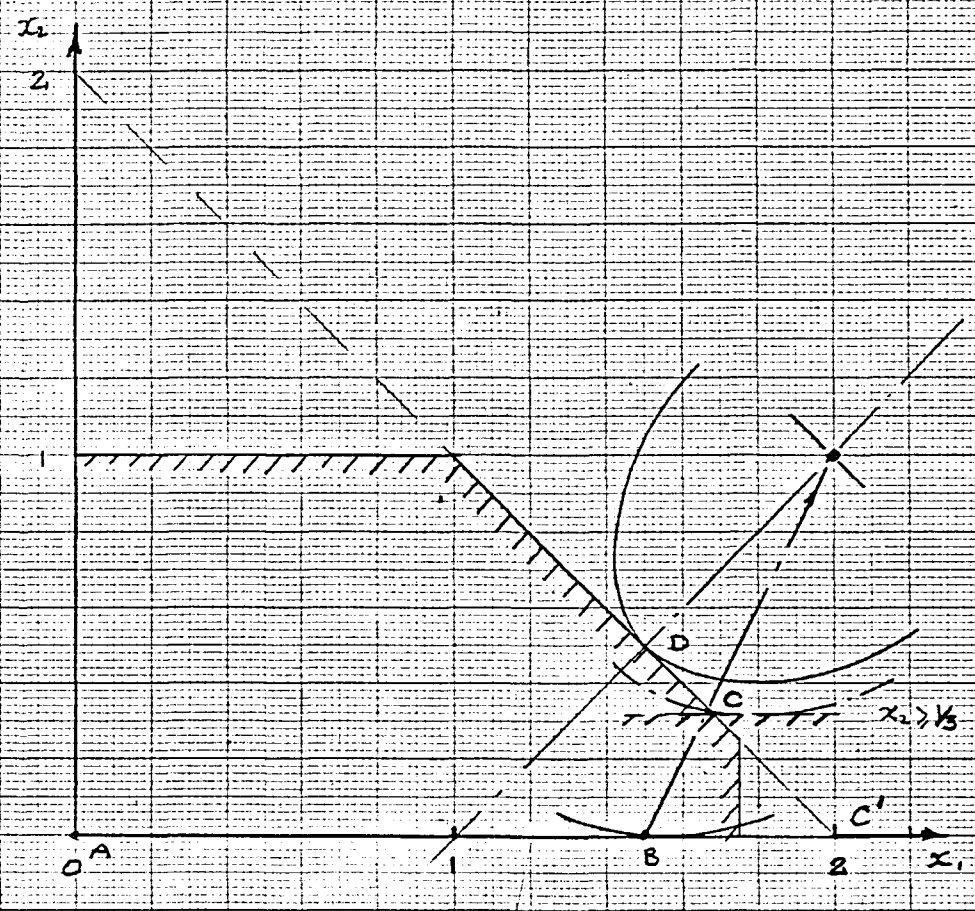


Fig. 5.1.

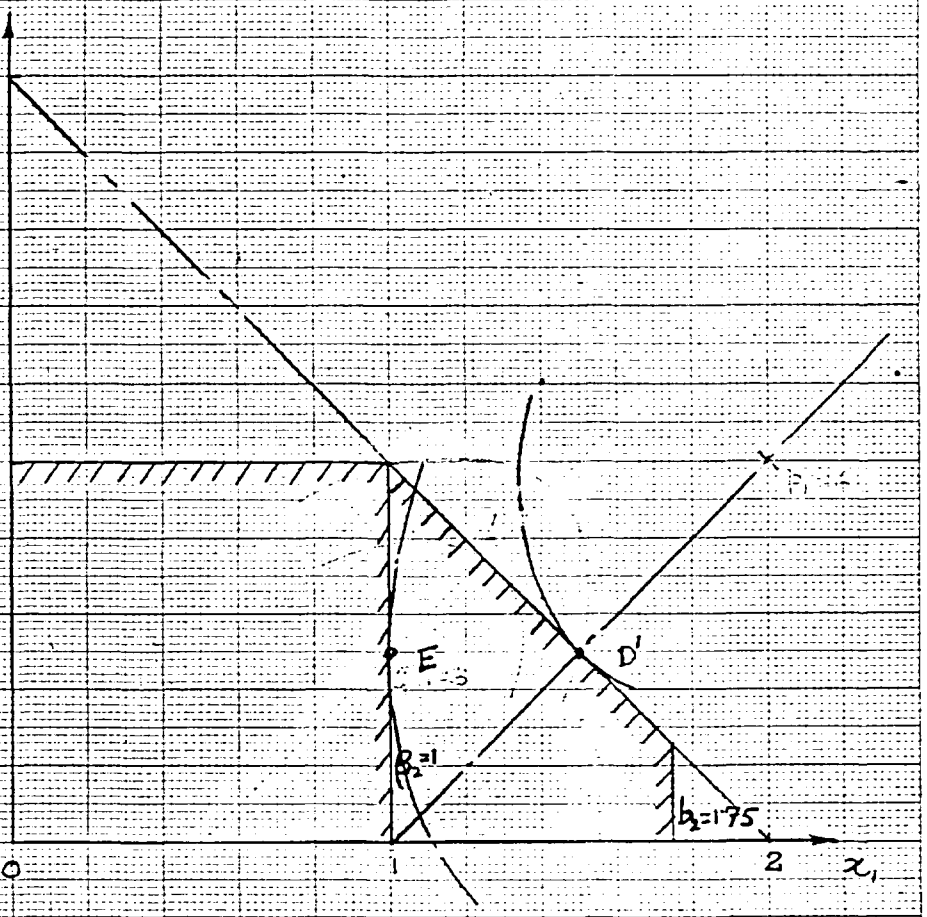


Fig. 5.2.

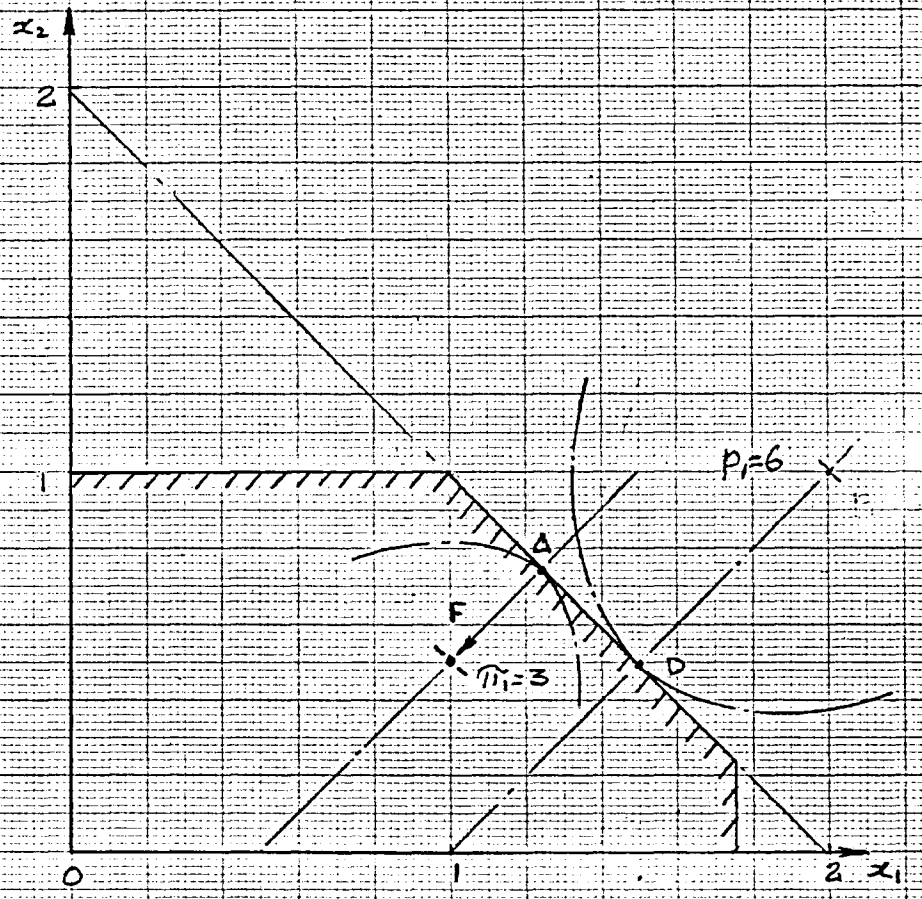

x_2
2
1
0
1
2
x_1
$p_1=6$
A
F
$\pi_1=3$
D

Fig. 5.3

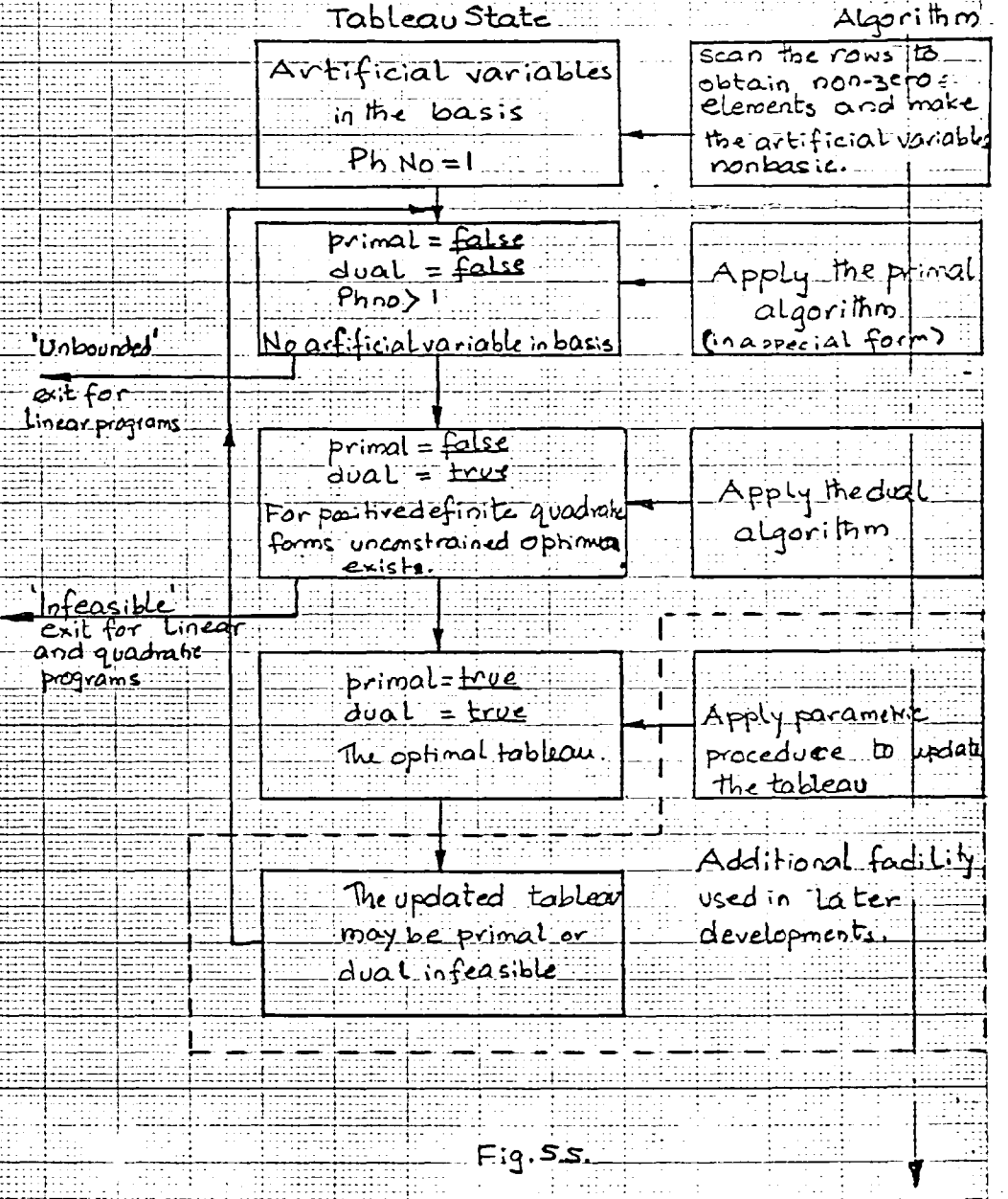


Tableau State
Algorithm
Artificial variables in the basis
Ph No = 1
scan the rows to obtain non-zero elements and make the artificial variable nonbasic.
primal = false
dual = false
Phno > 1
No artificial variable in basis
Apply the primal algorithm (in a special form)
'Unbounded'
exit for linear programs
primal = false
dual = true
For positive definite quadratic forms unconstrained optimum exists.
Apply the dual algorithm
'Infeasible'
exit for linear and quadratic programs
primal = true
dual = true
The optimal tableau.
Apply parametric procedure to update the tableau
The updated tableau may be primal or dual infeasible
Additional facility used in later developments.

Fig. 5.5

(supplementary material) containing the specifications for the library program.

f. On the positive definiteness of quadratic forms and Gauss's algorithm

A symmetric matrix C and its associated quadratic form x' C x is said to be positive definite if for all x, $(x \neq 0)$

$$x' C x > 0. \tag{5.28}$$

[If the strict inequality is replaced by the sign ≥ then this is a condition for positive semi-definiteness]. There are various ways of testing for positive-definiteness; from the computational point of view one is interested in obtaining this information with minimum number of elementary arithmetical operations.

The problem may be posed as an eigenvalue problem, [2.22] since, if all the eigenvalues of the matrix C are positive then the matrix C is positive definite. To determine the eigenvalues explicitly is computationally wasteful since it is only the signs of the eigenvalues which are of interest. However, one can perform the preliminary steps in this direction, namely, tridiagonalize the form C, [2.22] by applying Householder's algorithm, and obtain the form T, Fig. 5.6. Having obtained the matrix T, one can use the Sturm sequence [2.22] property to ascertain the change in sign of the eigenvalues.

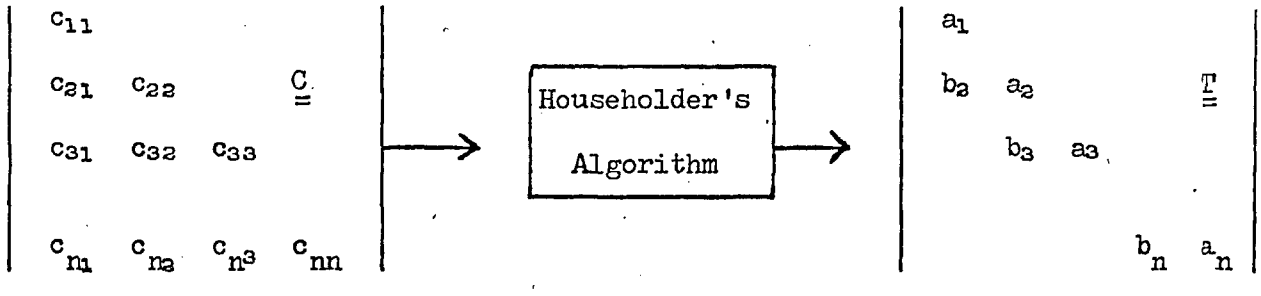


Fig. 5.6

Let the number of changes in sign in the sequence,

$$\left.\begin{aligned} p_0(\lambda) &= 1 \\ p_1(\lambda) &= a_1 - \lambda \\ p_r(\lambda) &= (a_r - \lambda)p_{r-1}(\lambda) - b_r^2\, p_{r-2}(\lambda) \end{aligned}\right\} \quad r=2(1)n \qquad (5.29)$$

be denoted by $v(\lambda)$. If $v(\lambda)= 0$, there are no roots less than λ. Thus if $v(\epsilon)= 0$, $\epsilon > 0$ then the matrix is positive definite and this can be used as a test for positive definiteness. A program was written in Algol which used a library procedure for Householder tridiagonalization, and the Sturm Sequence test was incorporated in this. The criterion for diagonal dominance which follows from the Hadamard-Greschgorin Theorem, was also tested at each intermediate step of the transformation.

If C' denotes the matrix at any intermediate step of tridiagonalization, then

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} | c'_{ij} | \leq c'_{ii} \quad , \quad i= 1,2,\ldots n \qquad (5.30)$$

expresses the condition for diagonal dominance.

Further studies in quadratic forms revealed that the Gauss's method for the diagonalization of a quadratic form is a more direct way of testing for the condition (5.28). This method is also a natural method, as it solves the normal equations associated with a quadratic form. It has been rediscovered by various workers [2.16], [2.20], and Beale [2.5] uses it to represent the value of the objective function. Consider the linear forms, (for the purpose of clarity an actual 3×3 system is considered)

$$\left.\begin{aligned} u_1 &= c_{11}\, x_1 + c_{12}\, x_2 + c_{13}\, x_3 \\ u_2 &= c_{21}\, x_1 + c_{22}\, x_2 + c_{23}\, x_3 \\ u_3 &= c_{31}\, x_1 + c_{32}\, x_2 + c_{33}\, x_3 \end{aligned}\right\} \qquad (5.31)$$

where the matrix C, is symmetric. The associated quadratic form $f(x)$ may

be expressed as,

$$f^3(x) = u_1 x_1 + u_2 x_2 + u_3 x_3 = c_{11} x_1^2 + 2c_{21} x_1 x_2 + c_{22} x_2^2 + 2c_{31} x_1 x_3 + 2c_{32} x_2 x_3 + c_{33} x_3^2 \qquad (5.32)$$

Let this be set up in the tableau format, Tableau 5.10, the symmetric matrix be written in full. Substituting, $x_2 = x_3 = 0$ and $x_1 = 1$ one

	$-x_1$	$-x_2$	$-x_3$	
u_1	$-c_{11}$	$-c_{12}$	$-c_{13}$	
u_2	$-c_{21}$	$-c_{22}$	$-c_{23}$	$\equiv -C^3$
u_3	$-c_{31}$	$-c_{32}$	$-c_{33}$	

Tableau 5.10

finds $f^3(x) = c_{11}$ and this must be positive to satisfy (5.28). Let a pivot operation be performed by the rules given in Appendix 5 on the element c_{11}. One obtains the tableau 5.11 with its corresponding skew symmetry/ symmetry property. In terms of the new tableau elements the quadratic

	$-u_1$	$-x_2$	$-x_3$	
x_1	$-c'_{11}$	c'_{21}	c'_{31}	
u_2	$-c'_{21}$	$-c'_{22}$	$-c'_{32}$	$\equiv -C^2$
u_3	$-c'_{31}$	$-c'_{32}$	$-c'_{33}$	

Tableau 5.11

form can be written out as,

$$
\begin{aligned}
f^3(x) &= u_1\ x_1 + u_2\ x_2 + u_3\ x_3 \\
&= u_1\ (c'_{11}\ u_1 - c'_{21}\ x_2 - c'_{31}\ x_3) \\
&\quad + (c'_{21}\ u_1 + c'_{22}\ x_2 + c'_{32}\ x_3)\ x_2 \\
&\quad + (c'_{31}\ u_1 + c'_{32}\ x_2 + c'_{33}\ x_3)\ x_3 \\
&= c'_{11}\ u_1^2 + c'_{22}\ x_2^2 \\
&\quad + 2\ c'_{32}\ x_3\ x_2 + c'_{33}\ x_3^2 \\
&= c'_{11}\ u_1^2 + f^2(x)
\end{aligned}
\tag{5.33}
$$

where $f^2(x)$ is the value of the quadratic form associated with the symmetric matrix c^2 of one dimension fewer,

$$
f^2(x) = (x_2\ ,\ x_3) \begin{vmatrix} c'_{22} & c'_{32} \\ c'_{32} & c'_{33} \end{vmatrix} \begin{pmatrix} x_2 \\ x_3 \end{pmatrix} \tag{5.34}
$$

For any value of x_2, x_3, one can always find a value of x_1 such that $u_1 = 0$. For such values of x_1 and u_1 it must be true (5.33) that $f^3(x) = f^2(x)$. It has therefore been argued, Stiefel [2.16], that for $f^3(x)$ to be greater than zero, $f^2(x)$ must be greater than zero. For C^2 associated with $f^2(x)$ this implies that $c'_{22} > 0$. The same argument can be extended to the next system $C' = c'_{33}$, which must be positive to make $f'(x) = x_3\ c'_{33}\ x_3$ positive. Thus performing Gaussian pivot operations on the successive tableaus down the diagonal elements, if at any stage the leading diagonal of the subsystem, that is the current pivot, turns out to be non negative (in the present Tucker representation), or non positive in the usual representation, then the matrix C is not positive definite. In connection with the positive definiteness of the matrices and the Gaussian algorithm, following points should be noted,

a) If a matrix is positive definite so is its inverse [2.16].

b) Gauss's algorithm when performed upto the nth step diagonalizes the quadratic form to a sum of squares of some other variables, in fact u variables, which are not necessarily orthogonal. This diagonalization follows from (5.33).

c) Beale's method of quadratic programming which can find a local optimum in a non convex problem, employs Gauss's algorithm for the representation of the quadratic form.

Gauss's algorithm provides the unconstrained stationary point in n-pivot steps and finds out if this is a maximum or a saddle point. For instance consider the quadratic form,

$$f(x) = (1,\ 3,\ 2) \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} - \tfrac{1}{2}\ (x_1,\ x_2,\ x_3) \begin{vmatrix} 5 & 1 & 10 \\ 2 & 1 & 3 \\ 10 & 3 & 21[38] \end{vmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \qquad (5.35)$$

where the C matrix has been taken from a subsystem given in [2.20] and an alternative value $c_{33} = 38$ is also considered. f(x) has a stationary point where the three partial derivatives $\frac{\partial f}{\partial x_1}$, $\frac{\partial f}{\partial x_2}$, $\frac{\partial f}{\partial x_3}$, vanish simultaneously. Such a point is obtained by solving the normal equations associated with the system (5.35). Let this be set up in the symmetric tableau format Tableau 5.12, where the elements are now defined for i and j= 0. Pivoting on $-c_{11} = -5$ one obtains the tableau 5.13. Pivoting on the elements $-c'_{22} = -\frac{1}{5}$ of tableau 5.13 one obtains the tableau 5.14. In this tableau $-c''_{33} = 4 > 0$, hence, the original matrix C is not positive definite. If one pivots on $-c''_{33} = 4$, one obtains the stationary point given by the zero column of the tableau 5.15 and this must be a saddle point. Considering the form corresponding to $c_{33} = 38$, one finds $-c''_{33} = -13$ in tableau (5.14)

hence the original matrix must be positive definite. Pivoting on this element the unconstrained maximum of the form f(x) is obtained and twice the value of the solution is contained in the zero of the tableau 5.16.

	x_0	$-x_1$	$-x_2$	$-x_3$
$2f(x) = u_0$	0			
u_1	-1	(-5)		
u_2	-3	-2	-1	
u_3	-2	-10	-3	-21[-38]

Tableau 5.12

	x_0	$-u_1$	$-x_2$	$-x_3$
$2f(x) = u_0$	$\frac{1}{5}$			
x_1	$\frac{1}{5}$	$-\frac{1}{5}$		
u_2	$-\frac{13}{5}$	$-\frac{2}{5}$	$(-\frac{1}{5})$	
u_3	0	-2	1	-1[-18]

Tableau 5.13

	x_0	$-u_1$	$-u_2$	$-x_3$
$2f(x) = u_0$	34			
x_1	-5	-1		
x_2	13	2	-5	
u_3	-13	-4	5	(4) [-13]

Tableau 5.14

	x_0	$-u_1$	$-u_2$	$-u_3$
$2f(x) = u_0$	$-\frac{33}{4}$			
x_1	18	3		
x_2	$-\frac{13}{4}$	-3	$\frac{5}{4}$	
x_3	$-\frac{13}{4}$	-1	$\frac{5}{4}$	$\frac{1}{4}$

Tableau 5.15

	x_0	$-u_1$	$-u_2$	$-u_3$
$2f(x) = u_0$	47			
x_1	-9	$-\frac{29}{13}$		
x_2	18	$\frac{33}{13}$	$-\frac{90}{13}$	
x_3	1	$\frac{4}{13}$	$-\frac{5}{13}$	$\frac{1}{13}$

Tableau 5.16

Chapter 6

Application of Quadratic Programming

Introduction

Quadratic programming has not been applied as widely as it deserves to be. The deterministic models normally proposed as linear programs, 'have an uncompromising addiction to basic solutions, ... these cannot be used to estimate "safety margins" which often have to be provided in practical situations to guard against uncertainities of one sort or other', (Beale, [2.25]). Stochastic linear programming [2.25], [2.4], and the E-V models ([2.17]), all tend to get round this oddity by attempting to minimize some relevant risks. In Section a of this chapter some of the more common applications of quadratic programming have been outlined, and in Section b a few general points regarding the application of quadratic programming and the usefulness of E-V diagrams are considered. Section c discusses the formulation and the solution of the optimum design of the vertical profile of highways by the application of quadratic programming.

a. Some more common applications of quadratic programming

The important applications of quadratic programming have been considered by Dantzig [2.4], Wolfe [2.11], Boot [2.13]. The following problems

a) regression analysis with non-negativity constraints and/or constraints of upper and lower bounds on the regression parameters,

b) to find the minimum of a general convex function subject to linear inequalities, where the convex function is being approximated by a positive definite quadratic form,

c) production planning by maximizing profit when the marginal cost is linearly varying, and the production

function is also considered to be linearly varying,

d) in a linear model, for given ranges of expected profits, to find solutions which minimize the variances (risk levels) of such profits,

admit formulation in the format of convex quadratic programming. Beale, [2.25], has suggested an interesting approach of turning a stoachastic linear programming problem into a quadratic programming problem.

The quadratic programming code described in the previous chapter was first applied to a problem of cell multiplication in medical biology. This involved minimization of a function of the form

$$f(p) = \sum_{j=1}^{n} a_j p_j + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} p_i p_j$$

subject to the constraints

$$\sum_{j=1}^{n} p_j = 1 \qquad (6.1a)$$

$$\text{or} \quad \sum_{j=1}^{n} p_j \leq 1 \qquad (6.1b)$$

and $p_j \geq 0$, $j = 1,2,3,\ldots n$, where the symmetric matrix C of elements c_{ij} is positive definite, and p_j's are certain probabilities to be estimated. The problem was formulated by Mr Peter Smith of Medical Research Council, and a complete description of the model is beyond the scope of the current work. However, the nature of the problem is interesting in that it is typical of many allocation problems. Because of the presence of cross terms in the quadratic form explicit treatment of the problem by dynamic programming is not possible. This difficulty can be circumvented by introducing new variables, that is actually diagonalizing the quadratic form, and re-writing the constraints in accordance with such transformations. Such a computing effort is equivalent to solving the problem by quadratic programming and is a good illustration of its power.

b. Studying the expected profits (E) and the minimum variances (V) of a linear model.

The pioneering work in this field is due to Markowitz [2.17]. The problem investigated by him was to select an efficient portfolio out of a list of securities. Whenever one considers a linear measure of performance, where the coefficients μ_j of such a linear form are estimated from observations.

one also obtains with these estimates a variance/covariance matrix σ of elements σ_{ij}. It can be said that a solution vector x of chosen activities which has a reward $E = \mu' x$, has also a risk level associated with it given by the quantity $V = x' \sigma x$. For the appropriate ranges of expected profit E it is therefore most efficient to find x-vectors which minimize V, this minimization being carried out over the convex region,

$$Ax \leq b \qquad (6.2)$$

$$\mu' x \geq E, \quad x \geq 0$$

defining the solution space of the feasible choice of the vectors x. To implement this analysis one therefore requires a convex quadratic programming code; more conveniently a parametric version of it which allows one to study the effect of changing the constant term of an inequality constraint or a linear cost coefficient. Making use of such a code one obtains an E-V diagram of the form shown in Fig. 6.1, and the decision maker ideally chooses his operating point on this curve of efficient E-V combination. For every point on this curve there exists a corresponding x-vector; in practice only a few of these are tabulated, representing a few discrete points. One might alternatively plot a diagram with E and $\sqrt{V}$ as the two axes.

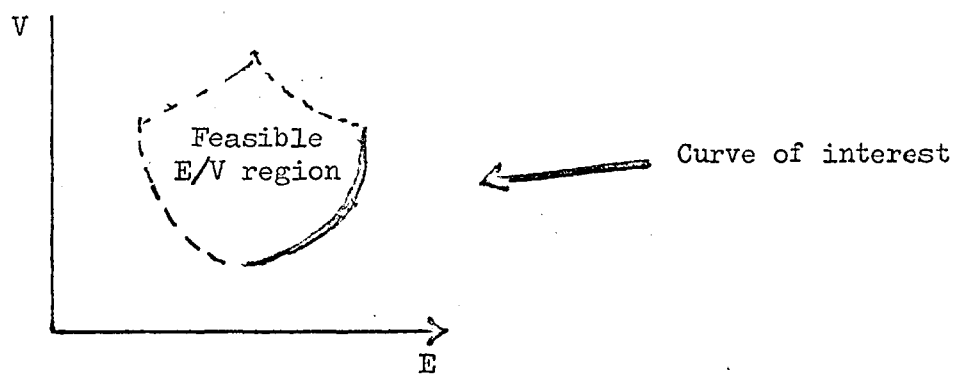


Fig. 6.1

Paine [2.18] has reported a very useful E-V analysis - a case study in portfolio selection, based on historical business data analysed on an aposteriori basis. The investments of a large finance company in forty

securities over a period of thirteen years were investigated. The time periods 1946-54 and 1955-59 represented two business cycles, over which the probabilities of the returns from each stock were governed by two different probability distributions. The E-V diagram for the first period was constructed and the actual investment policy was also represented in this diagram by the point F (Fig. 6.2). Fig. 6.3 shows how these efficient portfolios would have actually performed in the period 1955-59 compared with the investment decision followed by the finance company. The portfolios in the quadrant II are certainly inferior to F, those in

Diagram of efficient portfolios constructed from the data for 1946-54.

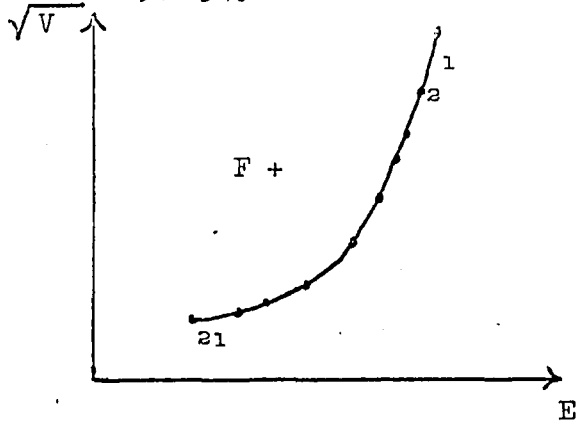


Fig. 6.2

E, $\sqrt{V}$ values of the efficient portfolios in the 1946-54 diagram plotted for 1955-59 conditions.

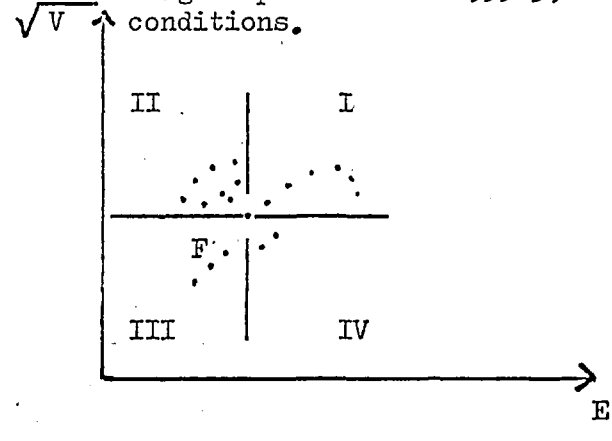


Fig. 6.3

the quadrants I and III certainly have points for and against such selections, and those in quadrant IV represent superior performance. Thus it has been concluded that if the probability distribution does not change, that is the investor feels that the future performance of the stocks will be the same as in the past, then one is justified in choosing portfolios which have been efficient in the past. If such an assumption cannot be justified, as shown in this example, then there is no guarantee that such an analysis on the past data will provide the best future portfolios.

The parametric quadratic program described in Chapter 5 has been employed for similar studies, a small example concerning the diversification

of resources in an advertising campaign and, more routinely, farm management problems proposed by the department of agricultural economics Wye College.

The problems in farm planning were formulated by Messrs Lawi and Megino of Wye College, and concerned the farm management decisions in Kenyan farms. The problems consisted in finding efficient vectors (like the efficient portfolios) of growing activities, that is, what to produce (coffee, maize, etc), and of hiring of labour at each month such that for the ranges of incomes (profits), E, the risk V was minimized. The constraints consisted of the balance equation for the land available, bounds on the growing activities, the balance equations for the fixed labour available in the twelve months of the year and the upper bounds on the hired labour. A typical problem, which was usually a subproblem of a larger model, consisted of 6 to 12 growing activities and 12 hiring activities, that is about 18 to 24 variables and 30 or more constraints. Only the variance figures for the estimated prices of the growing activities as obtained from the last ten years data were used in the model.

In the light of what has been mentioned by Paine [2.18] and discussed in the last paragraph, the users were advised to derive these variance figures and interpret their efficient farm management decisions accordingly.

c. Formulation and solution of the optimization of the vertical profiles of highways by quadratic programming.

The notations used in this section are defined as the problem is formulated. However, for ease of reference the more important symbols are given below.

Notation:

Let

l_j denote the chainage in units of length at a station j,

h_j denote the height in units of length at a station j,

measured from a datum,

k_j^t denote the unit cost (cost per unit volume) of cutting or filling between the stations j and j-1 as obtained in t^{th} iteration,

x_j denote the depth of cut or fill at the station j, that is the non-negative independent variables in which the problem is solved,

$v_j(x_{j-1}, x_j)$ denote the volume of earth work handled between the stations j and j-1,

α_j denote the area of a section of the road at the station j, the section being at right angles to the length of the road,

W_j denote the width of the road at the station j,

θ_j denote the angle of inclination at the station j of the side slope of the road with the horizontal,

Z_j denote the actual elevation of the alignment proposed at the station j, measured from the datum level,

G_j denote the maximum allowable gradient at the station j,

R_j^{sum} denote the minimum allowable radius of curvature at the station j of the summit curve; by convention it will be taken with a negative sign,

R_j^{val} denote the minimum allowable radius of curvature of the valley curve at the station j,

C denote set of indices of the stations which are specified to be in cut,

F denote set of indices of the stations which are specified to be in fill,

C^V denote the set of indices of the volumes which are in cut,

F^V denote the set of indices of the volumes which are in fill,

b denote an integer which numbers a borrow pit as a source,

s denote an integer which numbers a spoil heap as a sink,

V_i $i \in C^V$ denote the volumes of material available at the cut volumes of index i,

V_j $j \in F^V$ denote the volume of material required at the fill volumes of index j,

v_{ij} denote the volume of material transported between these two cut and fill volumes of indices i and j,

Π_{ij} denote the unit cost of moving the volume v_{ij} between the centres of gravity of these two volumes, V_i and V_j,

c_j denote the cost of purchasing unit area of land at the station j,

Given the horizontal alignment of a highway, and the survey data of the vertical profile along this route, one has to propose a vertical alignment which will respect the vertical curvature constraints at the summits and at the valley curves and the slope constraints, and such that the cost of hauling earthwork volumes from the points of cut and borrow pits to the points of fill and spoil heaps will be minimized. The volumes of earthwork are hauled by various mechanical movers and the cost of moving unit volume over a certain distance is a function of the distance (Fig. 6.4). The constraints of slope and curvature can be expressed as linear inequalities. If the unit costs of cutting and filling carried out at certain predetermined points, called stations, on the profile were available (note these are only governed by how material is carried from and to these locations), and if, making suitable assumptions, one could express the overall cost as a convex

function, then one could aim to obtain the global minimum of a limited problem. Introducing certain simplifying assumptions, which can none the less be well justified from the view-point of a design engineer, the very general problem can be split up into two linked subproblems. The first problem consists in making decisions as to how much to cut and how much to fill at certain stations on the profile, the subsequent problem is to determine how this excess and deficiency of earthwork volumes can be redistributed most efficiently. To be specific let these be called problem 1 and problem 2.

Problem 1:

Consider the terrain, Fig. 6.5, specified by a sequence of straight lines connecting the points (stations) (l_j, h_j) $j= 0,1,2,...,7$, l_j being the chainage and h_j the elevation from some convenient datum. The problem is to specify heights along the profile through which a road connecting A and B, the end points, should pass, thus defining the vertical alignment of the road. A design engineer makes some preliminary decision as to where to cut and where to fill material at these stations; this immediately defines two sets of indices,

C= 0,1,2,5, the stations where one is allowed to cut,

F= 3,4,6,7, the stations where one is allowed to fill.

Let a set of non-negative variables x_j, $j \in F$ or C, that is $j= 0,1,2,...7$, be introduced which specifies, in some convenient unit of length, how deep to cut or fill at these stations. If the alignment is required to pass through the end points, the corresponding variables x_0 and x_7 are constrained to be zero. Let k_j^o, $j= 1,2,...7$, denote the unit costs (in £s/cu yd.) of cutting or filling between the stations j and j-1, then the overall cost of obtaining the alignment specified by the set of variables x_j can be expressed as

$$\phi(x)= \sum_{j=1}^{7} k_j^o \, v_j(x_{j-1}, x_j), \qquad (6.3)$$

where $v_j\ (x_{j-1}, x_j)$ denotes the volume of cut or fill associated with the location j. It will be shown in what follows that the functions $v_j(x_{j-1}, x_j)$ are in fact quadratic. Consider the sectional view (Fig. 6.6)taken at chainage l_3 (t'-t", Fig. 6.5) Assuming the terrain to be level in the offset direction (i.e. at right angles to the length of the road), the end area of the road section will be given by the equation

$$\alpha_3 = W_3\ x_3 + x_3^2\ \cot\ \theta_3, \tag{6.4}$$

similarly the end area at the station 4 is given by,

$$\alpha_4 = W_4\ x_4 + x_4^2\ \cot\ \theta_4, \tag{6.5}$$

where W_3, W_4 are the widths of the flat tops (carriageway, verge) at these stations and θ_3, θ_4 the angles of the side slopes. The volume of earth-work between the stations 3 and 4 is therefore given by the equation

$$\begin{aligned} v_4(x_3, x_4) &= \tfrac{1}{2}(\alpha_3 + \alpha_4)\ 9(l_4 - l_3) \\ &= \tfrac{1}{2}(l_4 - l_3)\ (W_3\ x_3 + x_3^2\ \cot\ \theta_3 + W_4\ x_4 + x_4^2\ \mathrm{Cot}\ \theta_4) \\ &= p_3\ x_3 + q_3\ x_3^2 + p_4\ x_4 + q_4\ x_4^2. \end{aligned} \tag{6.6}$$

The volume of material to be cut between two adjacent stations which are specified to be in cut, can be similarly expressed as a quadratic in the variables associated with these two stations, say stations 1 and 2. However, for two neighbouring stations,one in cut and the other in fill,the corresponding volume is made up of a cut volume and a fill volume, and these are not strictly quadratic in the variables associated with these stations. To get round this difficulty a quadratic approximation is made to the sum of these two volumes, [2.26] and [2.27], Further, the error due to such an approximation may be reduced by having such change over points reasonably close. The objective function $\phi(x)$, (6.3), therefore reduces to a quadratic form

$$\phi(x) \sum_{j=1}^{7}\ k_j^{\circ}(p_{j-1} x_{j-1} + p_j\ x_j + q_{j-1} x_{j-1}^2 + q_j\ x_j^2). \tag{6.7}$$

Let μ_d denote the mean haulage distance for a typical profile and μ_c the corresponding cost. The initial guess or the estimate of the unit cost of cutting or filling may be expressed as

$$K_j^c = K_i^c = \mu_{c/s}, \text{ where } i \in C, \; j \in F. \tag{6.8}$$

One has to minimize $\phi(x)$ (6.7) subject to

(a) the slope constraint

$$\left.\begin{aligned} \frac{Z_j - Z_{j-1}}{l_j - l_{j-1}} &\leq G_j \\ \frac{-Z_j + Z_{j-1}}{l_j - l_{j-1}} &\leq G_j \end{aligned}\right\} \quad j = 1,2,\ldots 7 \tag{6.9}$$

where G_j is the maximum allowable gradient at station j, and

$$Z_j = h_j + x_j, \; j \in F \text{ or } Z_j = h_j - x_j, \; j \in C, \tag{6.10}$$

(b) the curvature constraint which can be expressed as, [2.26], [2.27],

$$\left.\begin{aligned} d_{j+1} Z_{j+1} + d_j Z_j + d_{j-1} Z_{j-1} &\geq -\left|\frac{1}{R_j}\right|^{sum} \\ -d_{j+1} Z_{j+1} - d_j Z_j - d_{j-1} Z_{j-1} &\leq \frac{1}{R_j}^{val} \end{aligned}\right\} \quad j = 1,2,\ldots 6 \tag{6.11}$$

where d_j are some appropriate linear multipliers of the dimension $(\text{length})^{-2}$, and R_j^{val}, R_j^{sum} are the minimum radii of curvature of the valley and the summit curves at station j, the radius of curvature of the summit curve being negative by convention. It is of interest to note that in the terminology of finite differences the expressions in the left hand side of (6.9) and (6.11) correspond to the first and second differences of the alignment curve that one is going to propose by satisfying these linear constraint inequalities. The constraints (6.9) and (6.11) can be expressed in terms of x_js (6.10) and in the example of Fig. 6.5 this leads to a set of 26 inequalities

$$\sum_{j=o}^{7} a_{ij} x_j \leq b_i; \; i = 1,2,\ldots(2\times7), \; (2\times7+1)\ldots2\times(7+6), \tag{6.12}$$

of which 14 are of the form

$$a_{ij} x_j + a_{ij+1} x_{j+1} \leq b_i$$

and 12 of the form

$$a_{ij-1}\, x_{j-1} + a_{ij}\, x_j + a_{ij+1}\, x_{j+1} \leq b_i.$$

This structuring is quite general. The problem 1, therefore, reduces to a convex quadratic programming problem, of minimizing the quadratic form (6.7) subject to the linear inequalities (6.12).

Problem 2:

One gross approximation in problem 1, is that the unit costs k_j^o are assumed to be known, and are represented by an overall mean value $\mu_{c/s}$. The cut and fill volumes obtained by the solution of problem 1 may be distributed optimally between the stations by means of a transportation algorithm and this will provide a more educated guess say k_j', $j \in C$ or F, regarding the unit costs associated with these stations j. Assume that a solution to problem 1 has been obtained and let Fig. 6.7 illustrate such an alignment. The profile is split up into a number of regions each designated for cut or fill and these serve as the sources and sinks respectively of the transportation problem. Let the areas representing these regions be numbered as shown in the Fig. 6.7. Further, let borrow pits and spoil heaps be introduced, which may be regarded as infinite sources and sinks. The problem can be set up in the transportation tableau format, (Tableau 6.1). Let

$C^V = 1,2,3,7,8,$ be the set of indices of the cut volumes,

and $F^V = 4,5,6,9,10,$ be the set of indices of the fill volumes.

Let the indices b and s denote the borrow pit and spoil heap respectively.

The associated transportation problem may be stated as

$$\text{minimize} \sum_{i \in C^V \cup b} \sum_{j \in F^V \cup s} v_{ij}\, \Pi_{ij} \qquad (6.13)$$

subject to the non-negativity conditions on the transported volumes v_{ij} and the material balance equations

$$\left.\begin{aligned} \sum_{j \in F^V \cup s} v_{ij} &= V_i \,, \quad i \in C^V \cup b, \\ \sum_{i \in C^V \cup b} v_{ij} &= V_j \,, \quad j \in F^V \cup s \end{aligned}\right\} \tag{6.14}$$

where V_i and V_j are obtained by solving problem 1. V_b and V_s , the quantities one can borrow and spoil, are set equal to very large amounts, and further

$$\sum_{i \in C^V \cup b} V_i = \sum_{j \in F^V \cup s} V_j \tag{6.15}$$

The coefficient π_{bs} is set equal to zero, π_{ij} , $i \in C^V$ $j \in F^V$ are obtained by determining the distance between the centres of gravity of the ith and the jth volume and reading off the corresponding cost from the cost curve Fig. 6.4. The costs Π_{ij}, i= b or j= s may be similarly obtained depending upon the physical distances of the borrow pit and the spoil heap and the actual costs of acquiring such earthwork volume or spoiling it.

$\Pi_{1,4}$	$\Pi_{1,5}$	$\Pi_{1,6}$	$\Pi_{1,9}$	$\Pi_{1,10}$	$\Pi_{1,s}$	V_1
$\Pi_{2,4}$	$\Pi_{2,5}$	$\Pi_{2,6}$	$\Pi_{2,9}$	$\Pi_{2,10}$	$\Pi_{2,s}$	V_2
$\Pi_{3,4}$	$\Pi_{3,5}$	$\Pi_{3,6}$	$\Pi_{3,9}$	$\Pi_{3,10}$	$\Pi_{3,s}$	V_3
$\Pi_{7,4}$	$\Pi_{7,5}$	$\Pi_{7,6}$	$\Pi_{7,9}$	$\Pi_{7,10}$	$\Pi_{7,s}$	V_7
$\Pi_{8,4}$	$\Pi_{8,5}$	$\Pi_{8,6}$	$\Pi_{8,9}$	$\Pi_{8,10}$	$\Pi_{8,s}$	V_8
Π_{b4}	Π_{b5}	Π_{b6}	Π_{b9}	Π_{b10}	0	V_b
V_4	V_5	V_6	V_9	V_{10}	V_s	

From

To

Tableau 6.1

Let v^0_{ij} denote the solution to the problem 2, that is the transportation problem (6.13), (6.14), (6.15) as displayed in Tableau 6.1. The cost of moving a volume of material to or from any of the ten volumes may be approximated by the quantities,

$$\left.\begin{aligned} \phi^0_j &= \tfrac{1}{2} \sum_{i \in C^V} \Pi_{ij}\, v^0_{ij} + v^0_{bj};\quad j \in F^V \\ \text{and}\quad \phi^0_i &= \tfrac{1}{2} \sum_{j \in F^V} \Pi_{ij}\, v^0_{ij} + v^0_{is};\quad i \in C^V \end{aligned}\right\} \qquad (6.16)$$

At the tth iteration the superscript '0' will be replaced by 't'.
Since cut and fill stations must have change-over points one finds that the sets C and F of the problem 1 are different from the sets C^V and F^V of the problem 2. In general, when two adjacent stations, j-1, j, are in cut, one can always match a pair of indices

$$k^t_{\substack{j \\ j \in C}} = \phi^t_{\substack{p/V_p \\ p \in C^V}}, \qquad \text{for instance } j=2,\ p=2; \qquad (6.17)$$

similarly, for two adjacent stations i-1 and i, in fill, one can match a pair of indices

$$k^t_{\substack{i \\ i \in F}} = \phi^t_{\substack{s/V_\gamma \\ \gamma \in F^V}}, \qquad \text{for instance } i=4,\ \gamma=5. \qquad (6.18)$$

When change-overs occur some inaccuracy in the unit costs are again introduced, [2.26] and an average value of the ϕs for the intermediate volumes are taken. The information regarding the unit costs as obtained by solving the problem 2, is used to solve problem 1 and the process is iterated. No stopping rule (based on any mathematical criterion) has been proposed; but when the user feels an acceptable, and in his judgement a good solution has been obtained, the process is terminated.

It has not been proved, but it turned out in most of the cases that the cost of producing the alignments always improved at each iteration, where the cost was taken as that provided by the solution of problem 2. This it may be claimed is logical to expect, since the heuristic employed is

the same as the manual design procedure, however, each subproblem is solved more accurately than one could otherwise do so. The solutions proposed by this method have proved to be always more economical than the manual designs and in one test case [2.26] than a computer solution proposed by the Scandinavian Road Federation. The typical size of the problems tackled consisted of 25 to 30 x variables, and 100 to 120 constraints for problem 1, representing about 3 to 4 km of a highway. The quadratic programming code solved these problems in about 3 to 4 minutes of computing time on the ICT Atlas 1 computer using about 24 k of fast store which included the storage of the program. The input to the problem 1 and the linking between the problems had to be carried out by having flexible matrix generator and data processing codes. The problem 2, was of course solved by standard transportation algorithm. Solving the transportation problem, data processing and the final polynomial smoothing [2.27] of the solutions constitute a much smaller computing effort compared to the main quadratic program which actually produces the alignment.

Finally, it is worth pointing out that the cost function of the problem 1, in fact contains further details which were not mentioned in the simplified description provided earlier, but these details in fact have been catered for in the program. For instance, one further considers a land acquisition cost between the stations j and j-1,

$$L_j = c_j(l_j - l_{j-1})\ (x_j \cot \theta_j + x_{j-1} \cot \theta_{j-1}) \qquad (6.19)$$

(where c_j is the cost per unit area of land to be acquired). This introduces a linear cost term. The gradient at any station j further introduces an user cost, which when summed over the appropriate distance and the planning horizon, may be expressed as

$$U_j = \text{constant term} + (\text{Gradient between } j \text{ and } j\text{-}1)^2\ (l_j - l_{j-1})$$

$$= \text{constant term} + \tau_{j-1} x_{j-1} + \tau_j x_j + K_j x_j^2 + K_{j-1} x_{j-1}^2 + \delta_{j,j-1} x_j x_{j-1}$$

(6.20)

The introduction of the user cost (6.20) leads to subdiagonal elements $\delta_{j,j-1}$ in the quadratic form to be considered in place of the form (6.7). The quadratic form which is no longer diagonal is always checked for positive definiteness; Hadamard-Greschgorin test for diagonal dominance, Chapter 5, Section f, has always proved to be adequate for this purpose. Extensive tests are being carried out at present to find out the effects of incorporating these user costs in the model ([2.27]).

In conclusion it may be said that the application of quadratic programming to road design is new; it is also pertinent, since it takes into account the costs of creating side slopes, which would be neglected if linear programming is used. It further enables one to study how the user costs are related to the actual cost of constructing the road.

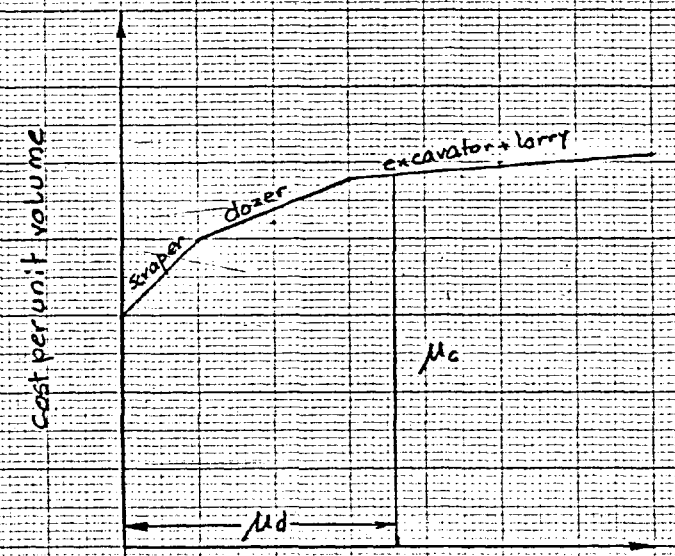


Fig. 6.4

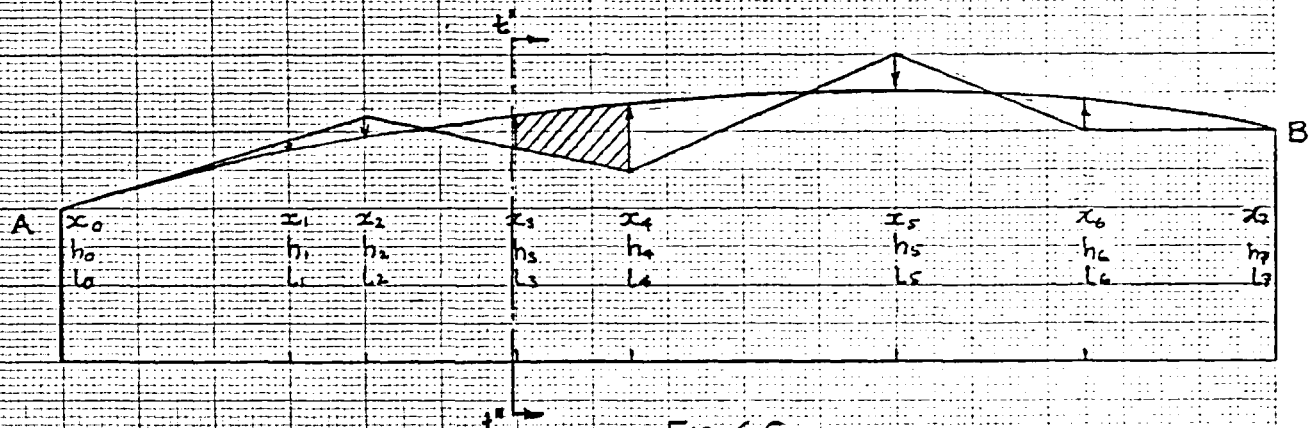


Fig. 6.5

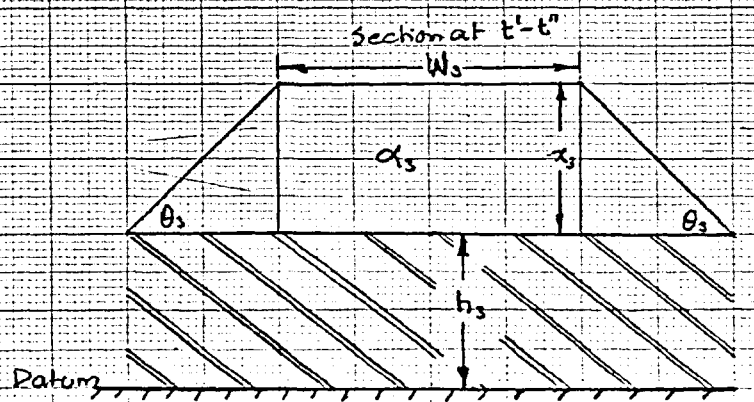


Fig. 6.6.

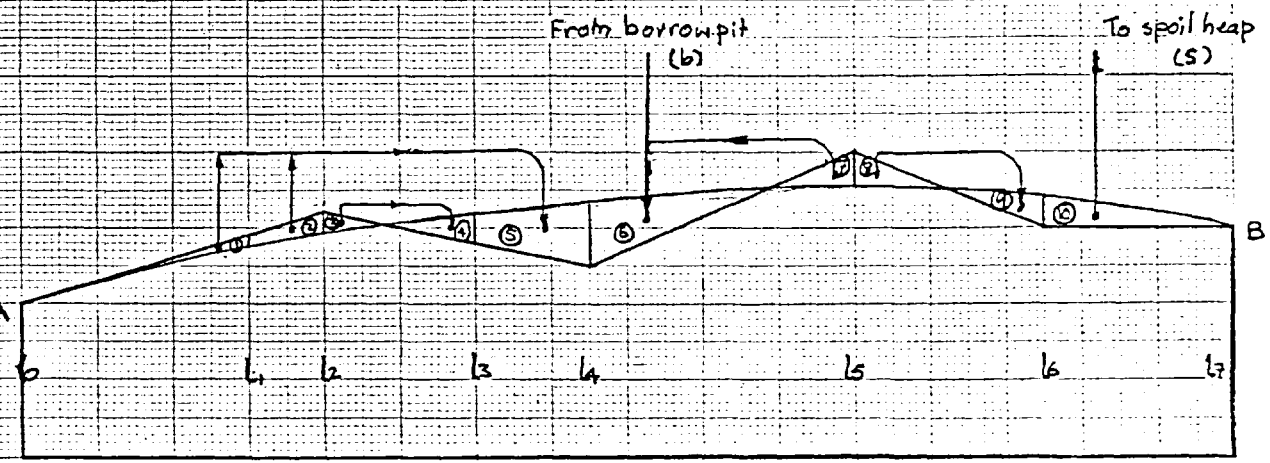


Fig. 6.7

PART 2

References

2.1 Hadley, G., Linear Programming, Addison-Wesley Publishing Co. Inc. (1962).

2.2 - , Non-linear and Dynamic Programming, Addison-Wesley and Co. Inc., (1965).

2.3 Simmonard, M., Linear Programming, (Translated by Jewell, W.S.,) Prentice Hall, (1966).

2.4 Dantzig, G.B., Linear Programming and Extensions, Princeton University Press (1963).

2.5 Beale, E.M.L., Mathematical Programming In Practice, Pitman, (1968).

2.6 Gomory, R.E., An Algorithm for Integer Solutions to Linear Programs, Recent Advances In Mathematical Programming, ed, Graves, R.L., and Wolfe, P., McGrawHill, (1963) 269-302.

2.7 - , An All-integer Integer Programming Algorithm, Industrial Scheduling, Ed. Muth, J.F., and Thompson, G.L., Prentice Hall (1963), 193-206.

2.8 - , On the Relation Between Integer and Non-integer Solutions to Linear Programs, Proceedings of National Academy of Sciences. (U.S.) 53, (1965), 260-265.

2.9 Thompson, G.L., The Stopped Simplex Method - 1 Basic Theory for Mixed Integer Programming, Integer Programming. Revue Francaise de Recherche Operationelle, 8 (1964), 159-182.

2.10 Finkelshteyn, Yu. Yu., Additional Restrictions for Problems of Integer Linear Programming, Engineering Cybernetics, (Translated

from Russian), May - June (1965), 38-39.

2.11 Wolfe, P., The Simplex Method for Quadratic Programming, Econometrica, 27 (1959), 382-398.

2.12 Beale, E.M.L., On Quadratic Programming, Naval Research Logistics Quarterly, 6 (1959), 227-243.

2.13 Boot, J.C.G., Quadratic Programming, North Holland Publishing Co. (1964).

2.14 Van-de-Panne, C., and Whinston, A., The Simplex and the Dual Method for Quadratic Programming, Operational Research Quarterly, 15 (1964), 355-388.

2.15 - and - , The Symmetric Formulation of the Simplex Method for Quadratic Programming, University of California, Los-Angeles, Western Management Science Institute, Working paper no. 112 December (1966).

2.16 Stiefel, E.L., An Introduction to Numerical Mathematics, Academic Press (1963).

2.17 Markowitz, H., The Optimization of a Quadratic Function Subject to Linear Constraints, Naval Research and Logistics Quarterly, 3 (1956), 111-133.

2.18 Paine, N.R., A Case Study in Mathematical Programming of Portfolio Selections, Applied Statistics, 15 (1966), 24-36.

× 2.19 Tui, H., Concave Programming Under Linear Constraints, Doklady Akademic NAUK S.S.S.R. (AMS Translation) 5 (1964), 1437-1440.

2.20 Beigtler, C.S., and Wilde, D.J., Diagonalization of Quadratic Forms by Gauss Elimination, Management Science, 12, January (1966), 371-379.

× 2.21 Thompson, G.L., Tonge, F.M., Zionts, S., Techniques for Removing Non-binding Constraints and Extraneous variables from Linear Programming Problems, 12 (1966), 7, 588-608.

2.22 Wilkinson, J.H., The Algebraic Eigenvalue Problem, Clarendon Press Oxford (1965), 299-302.

2.23 Tucker, A.W., Combinatorial Theory Underlying Linear Programs, Recent Advances in Mathematical Programming, Ed., Graves, R.L., and Wolfe, P., McGrawHill, (1963), 1-16.

2.24 Balinski, M.L., and Gomory, R.E., A Mutual Primal Dual Simplex Method, Recent Advances in Mathematical Programming, Ed., Graves, R.L., and Wolfe, P., McGrawHill, (1963), 17-27.

2.25 Beale, E.M.L., The Use of Quadratic Programming in Stochastic Linear Programming, Rand Corporation Report p-2404 (1961).

2.26 Calogero, V., and Mitra, G., Optimization of the vertical Alignment of Highways by Quadratic Programming, internal report of the Institute of Computer Science, in preparation.

2.27 Calogero, V., Computer Aided Highway Design, PhD Thesis to be submitted.

2.28 Richards, D.B.C., 'Integer Programming Theory and Practice', PhD Thesis, University of London (1968).

PART 3

Tree Search Schemes and Heuristic Methods

Introduction

The problems considered in this part have the property that the solution space defined by the linear constraint equations is either an integer linear space (pure integer problems), or a space in which variables in some directions can assume integer values only, and in others can vary continuously (mixed integer problems). The Cutting plane method mentioned in Chapter 4 is one way of systematically exploring such a solution space. An alternative technique of investigating this type of solution space has become increasingly important in recent years and is known as the 'Branch and Bound' method.

For the solution of mixed integer linear programs Land and Doig [3.1] first introduced the concept of constructing a tree of subproblems and searching it in an ordered fashion. However, their approach has proved difficult to program. Little, Murty, Sweeney and Carel [3.2] first reported a very successful implementation of a tree search scheme to solve the travelling salesman problem. This they called the Branch and Bound method. The travelling salesman problem can be formulated as a pure integer programming problem and the close relationship has been later exploited by Dakin [3.3], Beale and Small [3.4], who have developed similar methods of constructing a bifurcating tree of subproblems to solve the general mixed integer linear programming problems. Driebeck [3.5] has also developed a tree search method for the mixed integer problems. A method due to Thompson [2.9] belongs to this class but is not exactly of this type. Burstall [3.6] has considered the scheme in the context of machine intelligence and has solved the cryptarithm problem by a tree search algorithm.

In general, one is interested in the growth of the tree mentioned

above and the rules or the heuristics of branching which determine this has been considered by all these workers. Little, et al [3.2] selected variables which made one of the branches most unattractive. They then investigated the problem in the other branch of the tree. Beale and Small [3.4] have a similar scheme. Burstall [3.6] first evaluates what he calls the 'residual space size'. This is the logarithm of the product of the ranges of the variables. The 'most unattractive variable' i.e., with the smallest space size for the sum of its adjacent problems is then chosen for branching. Dakin [3.6] mentions applying a dual algorithm and evaluating the variable which when branched on produces the largest change in the objective function. This he regards as the most promising of all the heuristics tried by him. Although, these heuristics uniquely define the growth of the relevant trees in none of these investigations is a comparative study made of the alternative branching heuristics. Some work has been done in this direction and is described in Chapter 9.

In what follows, the pertinent features of a general branch and bound scheme is introduced. To explore a mixed integer linear space the branch and bound method works on the principle that subproblems are proposed which introduce primal infeasibility. This is accomplished by setting integer bounds on the variables, the integrality of which are being examined. Starting from the continuous optimum solution (i.e. a solution arrived at by disregarding the integer constraints on all the variables), a tree of subproblems is generated and the subproblems at each node investigated. The problems down the branches of this tree have potentially decreasing optimal costs for the corresponding maximizing problems. If there exists at most one feasible solution, the search will of course enumerate this possibility. Any subproblem that has a cost less than the currently best feasible solution, is not considered any more i.e., the corresponding node deleted. The same is true for any subproblem that is infeasible (i.e. has no solution).

The search is terminated when the currently best feasible solution is proved to be optimal, as the tree of subproblems with lower cost is exhausted.

To tackle a problem by the branch and bound scheme it is desirable that

1) if the integrality conditions of the variables are ignored the corresponding problem is a convex mathematical programming problem,

2) a dual algorithm for solving the problem is at hand so that primal infeasibility can be efficiently resolved. This is not an essential prerequisite, since primal infeasibility can be dealt with, without explicit application of the dual algorithm. However, it is important that the duality theory can be applied to the problem, thus ensuring that the subproblems proposed have non increasing optimal costs along any branch of the tree. This is necessary for the convergence of the search. If the dual simplex method of using the parametric facility on the right hand side is one way of organizing the branching [3.5], then the other is typified by the bounded variable technique adopted by Dakin [3.3],

3) a technique (usually a stack mechanism) to store the tree of subproblems is available.

All these apply, or can be implemented for mixed integer quadratic programs and a technique has been developed, Chapter 7, to solve such problems.

The general scheme of branch and bound has been further extended, whereby either primal or dual infeasibilities can be introduced for branching. These are then resolved to solve the successor problems. This applies only to problems of special structure considered in Chapter 8 and Chapter 9. In the concluding chapter the close relationship between mathematical programming and the heuristic approach towards problem solving is considered. Heuristic search is popularly considered a provider of a good solution quickly, but which does not look for or guarantee an optimal solution. As opposed to this concept the heuristic technique is interpreted as one that arrives at a good solution quickly; further, when backed by a proper mathematical theory it can produce the global optimum by a finite search.

Chapter 7

A Branch and Bound Algorithm for Mixed Integer Quadratic Programs

Introduction

In this chapter a method is developed for the solution of the mixed integer quadratic programming problem. The problem is stated in Section a. In Section b the theory underlying the method is considered, and the convergence of the branching procedure adopted for the solution of the problem is discussed. In Section c, a small example is worked out and in Section d, the algorithm is formally stated. Section e discusses the computer implementation of the method and the solutions of a few test problems. In the knowledge of the author there exists no other technique for the solution of the mixed integer quadratic programming problem. A method due to Kunzi and Oettli [3.7] employs a cutting plane technique for the solution of integer quadratic programs, however, no computer implementation has been reported.

a. Statement of the problem

Consider the problem Q:

$$\text{Maximize } Q(x) = \sum_{j=1}^{n} p_j x_j - \tfrac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_i x_j \qquad (7.1)$$

subject to the constraints,

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \; ; \; i=1,2,\ldots m \qquad (7.2)$$

and $$x_j \geq 0, \; j=1,2,\ldots n \qquad (7.3)$$

and $$x_j \equiv 0 \bmod (1) \; , \; j \in I, \qquad (7.4)$$

where I is a set of indices of the variables which are constrained to be integral. Further, the symmetric matrix $C=[c_{ij}]$ of the quadratic form is assumed to be positive definite. The last assumption implies that if the integrality constraint (7.4) is ignored then the corresponding problem reduces to a convex quadratic programming problem. Let $s(Q)$

denote the solution space of the problem Q i.e. represent the constraints (7.2), (7.3), (7.4) and let Z_Q denote the cost of the optimal solution to the problem Q.

b. A branch and bound procedure

Let $\bar{Q}$ denote the convex problem mentioned in the last section, i.e. the problem Q with the integrality constraint (7.4) ignored and let $s(\bar{Q})$ denote the solution space of this problem, i.e. (7.2), (7.3) and $Z_{\bar{Q}}$ the cost of its optimal solution. Problem $\bar{Q}$ can be solved employing a convex quadratic programming algorithm, hapter 5, leading to the bound

$$Z_{\bar{Q}} \geq Z_Q \tag{7.5}$$

since, $s(Q) \subset s(\bar{Q})$.

Consider any variable $x_i^o = a_{io}^o \neq 0 \bmod(1)$, for $i \in I$ in the optimal solution x^o of the problem $\bar{Q}$. If one such x_i^o exists then the constraint (7.4) is not satisfied and x^o is not a feasible solution of the mixed integer problem Q. However, let the following two subproblems be defined

Q(ub):

Minimize Q(x) given in (7.1) (7.6)

subject to the constraints (7.2), (7.3), (7.4) and

$$x_i \leq [a_{io}^o] \tag{7.7}$$

and

Q(lb):

Minimize Q(x) given in (7.1) (7.8)

subject to the constraints (7.2), (7.3), (7.4) and

$$x_i \geq [a_{io}^o] + 1, \tag{7.9}$$

where the square bracket stands for the integer part of the number enclosed. Let s(Q(ub)), s(Q(lb)) represent the solution spaces of the above two problems respectively. Then fig. 7.1 illustrates how the solution spaces are related to each other.

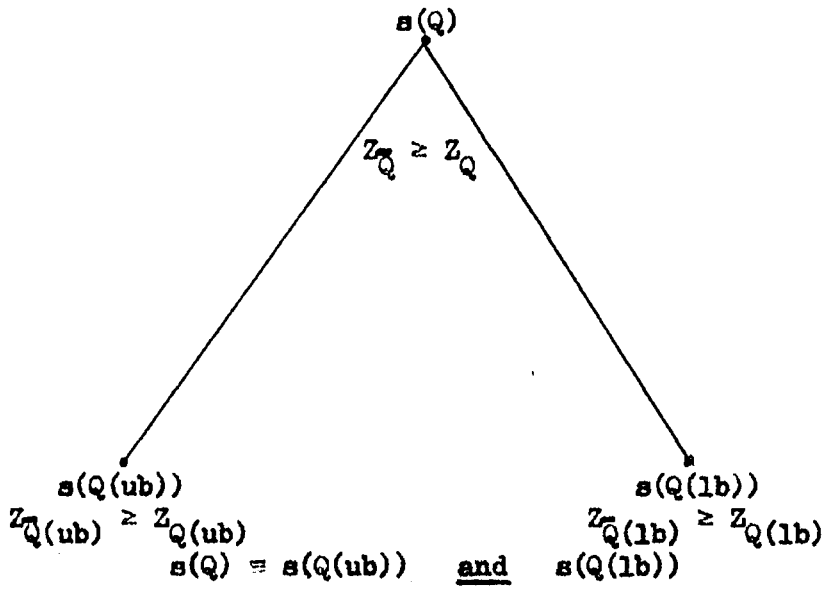


Fig. 7.1.

Let $s(\bar{Q}(ub))$, $s(\bar{Q}(lb))$, $Z_{\bar{Q}(ub)}$, $Z_{\bar{Q}(lb)}$, denote the solution spaces, and the costs of the optimal solutions of the corresponding $\bar{Q}(ub)$, $\bar{Q}(lb)$, problems with the integer constraints ignored. Further, let $Z_{Q(ub)}$, $Z_{Q(lb)}$ denote the costs of the optimal solutions to the Q(ub) and Q(lb) problems. Then the following bounds must hold (cf Fig. 7.1).

$$Z_{\bar{Q}(ub)} \geq Z_{Q(ub)} \ , \ Z_{\bar{Q}(lb)} \geq Z_{Q(lb)} \tag{7.10}$$

since, $s(Q(ub)) \subset s(\bar{Q}(ub))$, $s(Q(lb)) \subset s(\bar{Q}(lb))$

and $$Z_{\bar{Q}(ub)} \leq Z_{\bar{Q}} \ , \ Z_{\bar{Q}(lb)} \leq Z_{\bar{Q}} \tag{7.11}$$

since, $s(\bar{Q}(ub)) \subset s(\bar{Q})$ and $s(\bar{Q}(lb)) \subset s(\bar{Q})$.

Moreover by direct enumeration it must be true that

$$Z_Q = \max \{Z_{Q(ub)} \ , \ Z_{Q(lb)}\} . \tag{7.12}$$

The result of enumerating the original problem into two subproblems is that, the open interval $[a^o_{ip}] < x_i < [a^o_{ip}] + 1$ is excluded from the continuous problems which are solved to obtain better upper bounds $Z_{\bar{Q}(ub)}$, $Z_{\bar{Q}(lb)}$ on the original mixed integer problem. It may be noted that the problem is formulated as a maximizing problem. From the solution x^o of the problem $\bar{Q}$ one can solve by the application of the dual algorithm, the problems $\bar{Q}(ub)$ and $\bar{Q}(lb)$. From each of the two nodes similar subproblems may be proposed and the corresponding continuous problems solved for better

upper bounds on the cost of the optimal solution. The development of the tree leading to potentially decreasing local upper bounds to, the optimal costs of the subset of problems at any branch of the tree. Whenever the solution of the continuous problem at any node of the tree is feasible, i.e. (7.4) is satisfied, the cost of this solution is assessed to update the currently best solution. If the local upper bound(i.e. the cost of the optimal solution of the continuous problem at a node) becomes less than the cost of the currently best solution, or if the problem at any node becomes infeasible i.e. the constraints become inconsistent, then the branch leading to this node is deleted. The optimal solution if it exists, is obtained by the algorithm because,

a) the cost of the optimal solutions to the continuous problems in a path from the stem to any terminal node of the tree is potentially decreasing(7.11),

b) the possible ranges of the integer constrained variables are finite (consider $0 \leq x_i \leq M$, $i \in I$, and M a very large quantity), hence the number, and the growth, of the branches of the tree must be finite.

Fortunately, all the nodes of the tree do not need to be enumerated because of the facility of pruning the tree. The search is terminated when the tree is exhausted; the currently best solution is then the optimal solution, and if no feasible solution exists then problem Q must be infeasible. The complete algorithm is stated in Section d, however, it is important to note at this point, that the implementation of the algorithm hinges on the fact that duality theory is applicable to the continuous convex quadratic programming problem at any node of the tree. Therefore by the application of the dual algorithm, the convex subproblem at a successor node can be solved starting from the optimal tableau of the convex problem at the current node.

c. An example

In this section a small illustrative example is solved by hand computation. The symmetric tableau is in the same format as that developed in Chapter 5; the upper bounds are introduced as explicit constraints and the lower bounds are catered for by shifting the axes in the relevant directions by the amounts given by the lower bounds. Thus the tableau needs to be augmented only by the rows and columns of the integer constrained structural variables, and one needs a means of recording shifts in these axial directions, so that the solutions can be referred back to the original axes system. Out of the tree of subproblems developed and stored, the last stored subproblem is always extracted and investigated. In the subproblems the integrality of the variables is examined in the ascending order of the indices in the set I.

Consider the problem given by the data,

$$p' = [6,\ 0]\ ;\quad c_{ij}: \begin{bmatrix} 4 & \\ -2 & 4 \end{bmatrix}$$

$$a_{ij}: \begin{bmatrix} 5 & 6 \\ 2 & -1 \end{bmatrix}\ ;\quad b = \begin{bmatrix} 11 \\ 2 \end{bmatrix} \quad \text{and } I = \{1,2\}$$

and one has to maximize, (writing out (7.1), (7.2), (7.3), (7.4) in full)

$$Q(x) = 6x_1 - \tfrac{1}{2}\{4x_1^2 - 4x_1x_2 + 4x_2^2\}$$

subject to

$$5x_1 + 6x_2 \leq 11$$

$$2x_1 - x_2 \leq 2$$

$$x_1,\ x_2 \geq 0 \text{ and } x_1,\ x_2 \equiv 0 \bmod(1).$$

The above can be set up in the following tableau form and the corresponding $\bar{Q}$ problem can be solved applying the algorithm of Chapter 5.

Tableau 0

	x_0	$-x_1$	$-x_2$	$-u_3$	$-u_4$
u_0	0·0000				
u_1	-6·0000	-4·0000			
u_2	0·0000	2·0000	-4·0000		
x_3	11·0000	5·0000	6·0000	0·0000	
x_4	2·0000	2·0000	-1·0000	0·0000	0·0000

double pivot.

Tableau 1

	x_0	$-u_1$	$-x_2$	$-u_3$	$-x_4$
u_0	8·0000				
x_1	1·0000	0·0000			
u_2	-3·0000	0·5000	-3·0000		
x_3	6·0000	0·0000	8·5000	0·0000	
u_4	1·0000	-0·5000	0·0000	2·5000	-1·0000

double pivot

Tableau 2

	x_0	$-u_1$	$-u_2$	$-x_3$	$-x_4$
u_0	10·7405				
x_1	1·3529	0:0000			
x_2	0·7059	0·0000	0·0000		
u_3	0·1038	-0·0000	-0·1176	-0·0415	
u_4	0·7405	-0·3529	0·2941	0·1038	-1·2595

The continuous optimum solution to Q problem.

The integrality condition is violated by $x_1 = 1{\cdot}3529 \not\equiv 0 \bmod(1)$ and hence two subproblems are generated by augmenting the above tableau by the constraints $x_1 \leq 1$ and $x_1 \geq 2$ which are upper bound and lower bound constraints respectively. From tableau 3 onwards upper bound constraints are introduced explicitly such that x_5 and x_6 are the upper bound slack variables for x_1 and x_2. The rows in the tableau for these variables

x_5, x_6 can be constructed from the symmetry property of the tableau and from the fact that the entries in these rows are the same as those for rows x_1 and x_3 with the signs of the coefficients reversed. Of course, the corresponding dual variables u_5 and u_6 are similarly introduced. Initially the entries corresponding to the upper bound constraints are large positive numbers such that these are never violated. However, in the subproblems these are reduced to the appropriate integer upper bounds for the variables whose integrality are being investigated. The lower bounds are assumed zero initially, and then altered by positive integral amounts and noted next to the variable to the left. Tableau 3 is a representation of Tableau 2 with x_1, $x_2 \geq 0$ lower bounds and x_1, $x_2 \leq 1000$ upper bounds.

Tableau 3 ≡ Node 0

Lower bounds		x_0	$-u_1$	$-u_2$	$-x_3$	$-x_4$	$-u_5$	$-u_6$
	u_0	10·7405						
0	x_1	1·3529	0·0000					
0	x_2	0·7059	0·0000	0·0000				
	u_3	0·1038	0·0588	-0·1176	-0·0415			
	u_4	0·7405	-0·3529	0·2941	0·1038	-12595		
upper bounds								
1000	x_5	998·6471	0·0000	0·0000	-0·0588	-0·3529	0·0000	
1000	x_6	999·2941	0·0000	0·0000	-0·1176	0·2941	0·0000	0·0000

Tableau 4 ≡ Node 1

lower bounds		x_0	$-u_1$	$-u_2$	$-x_3$	$-x_4$	$-u_5$	$-u_6$
	u_0	10·7405						
2	x_1	-0·6471	0·0000	0·0000	0·0588	0·3529	0·0000	0·0000
0	x_2	0·7059	0·0000	0·0000	infeasible row			
	u_3	0·1038	-0·0588	-0·1176	-0·0415			
	u_4	0·7405	-0·3529	0·2941	0·1038	-1·2595		
upper bounds								
1000	x_5	998·6471	0·0000	0·0000	-0·0588	-0·3529	0·0000	
1000	x_6	999·2941	0·0000	0·0000	-0·1176	0·2941	0·0000	0·0000

The subproblem in the branch $x_1 \geq 2$, Tableau 4, Node 1 cf Fig. 7.2 is clearly infeasible. To illustrate this explicitly, construct the corresponding row of the Tableau 4 employing the symmetry property. One obtains lower bound = axial shift = 2 and $a^0_{10} = -0{\cdot}6471$, since all the entries in the row are non-negative this infeasibility cannot be resolved. The corresponding node is therefore deleted. Tableau 5 represents the problem in the other branch of the tree given by the

Tableau 5

lower bounds		x_0	$-u_1$	$-u_2$	$-x_3$	$-x_4$	$-u_5$	$-u_6$
	u_0	10·7405						
0	x_1	1·3529	0·0000					
0	x_2	0·7059	0·0000	0·0000				
	u_3	0·1038	-0·0588	-0·1176	(-0·4152) ←		single pivot	
	u_4	0·7405	-0·3529	0·2941	0·1038	-1·2595		
upper bounds								
1	x_5	-0·3529	0·0000	0·0000	-0·0588	-0·3529	0·0000	
1000	x_6	999·2941	0·0000	0·0000	-0·1176	0·2941	0·0000	0·0000

Tableau 6

lower bounds		x_0	$-u_1$	$-u_2$	$-u_3$	$-x_4$	$-u_5$	$-u_6$
	u_0	11·000						
0	x_1	1·5000	-0·0833					
0	x_2	1·0000	-0·1667	-0·3333				
	x_3	-2·5000	1·4167	2·8333	-24·0833			
	u_4	1·0000	-0·5000	0·0000	2·5000	(-1·0000) ←		
upper bounds								
1	x_5	-0·5000	0·0833	0·1667	-1·4167	-0·5000	-0·0833	
1000	x_6	999·0000	0·1667	0·3333	-2·8333	0·0000	-0·1667	-0·3333

constraint $x_1 \leq 1$; $x_5 = a^0_{50} = -0.3529$ is the amount by which this bound is violated and introduces primal infeasibility. Applying the dual algorithm one generates a sequence of tableaus until the optimal Tableau, 8 ≡ Node 2, is obtained. The integrality requirement of $x_8 = a_{80} = 0.50$ is violated and hence two new bounds are generated $x_8 \leq 0$ and $x_8 \geq 1$.

Tableau 7

lower bounds		x_0	$-u_1$	$-u_2$	$-u_3$	$-u_4$	$-u_5$	$-u_6$
	u_0	12.0000						
0	x_1	2.0000	-0.3333					
0	x_2	1.0000	-0.1667	-0.3333				
	x_3	-5.0000	2.6667	2.8333	-30.3333			
	x_4	-1.0000	0.5000	0.0000	-2.5000	-1.0000		
upper bounds								
1	x_5	-1.0000	0.3333	0.1667	-2.6667	-0.5000	-0.3333	←
1000	x_6	999.0000	0.1667	0.3333	-2.8333	0.0000	-0.1667	-0.3333

Tableau 8 ≡ Node 2

lower bounds		x_0	$-u_1$	$-u_2$	$-u_3$	$-u_4$	$-x_5$	$-u_6$
	u_0	9.0000						
0	x_1	1.0000	0.0000					
0	x_2	0.5000	0.0000	-0.2500				
	x_3	3.0000	0.0000	1.5000	-9.0000			
	x_4	0.5000	0.0000	-0.2500	1.5000	-0.2500		
upper bounds								
1	u_5	3.0000	-1.0000	-0.5000	8.0000	1.5000	-3.0000	
1000	x_6	999.5000	0.0000	0.2500	-1.5000	0.2500	-0.5000	-0.2500

Tableau 9

lower bounds		x_0	$-u_1$	$-u_2$	$-u_3$	$-u_4$	$-x_5$	$-u_6$
	u_0	9·0000						
0	x_1	1·0000	0·0000					
0	x_2	0·5000	0·0000	-0·2500				
	x_3	3·0000	0·0000	1·5000	-9·0000			
	x_4	0·5000	0·0000	-0·2500	1·5000	-0·2500		
upper bounds								
1	u_5	3·0000	-1·0000	-0·5000	8·0000	1·5000	-3·0000	
0	x_6	-0·5000	0·0000	0·2500	-1·5000	0·2500	-0·5000	-0·2500

Tableau 9 represents the former problem. Again applying the dual algorithm the optimal tableau 10 is generated, this also provides a feasible integer solution, u0= 8 i.e. the cost $^8/_2$= 4, x_1 = 1, x_2 = 0. Investigating the other branch $x_2 \geq 1$ tableau 11 is generated and the axial shift of 1 is noted

Tableau 10 ≡ Node 3

lower bounds		x_0	$-u_1$	$-u_2$	$-u_3$	$-u_4$	$-x_5$	$-x_6$
	u_0	8·0000						
0	x_1	1·0000	0·0000					
0	x_2	0·0000	0·0000	0·0000				
	x_3	6·0000	0·0000	0·0000	0·0000			
	x_4	0·0000	0·0000	0·0000	0·0000	0·0000		
upper bounds								
1	u_5	2·0000	-1·0000	0·0000	5·0000	2·0000	-4·0000	
0	u_6	2·0000	0·0000	-1·0000	6·0000	-1·0000	2·0000	-4·0000

with the consequent introduction of the primal infeasibility. The optimal tableau 12 ≡ Node 4 is produced in one pivot step. It satisfies the integrality constraint and is of cost = $^{u0}/_2$= $^8/_2$= 4 ; Referring back to the original axes system the solution is given by x_1 = 1, x_2 = 0 + 1= 1 (u2 in the basis implies x_2'= 0, and x_2 = x_2' + 1= 0 + 1). Fig. 7.2 is intended to provide a geometrical interpretation of adding upper and lower

Tableau 11

lower bounds		x_0	$-u_1$	$-u_2$	$-u_3$	$-u_4$	$-x_5$	$-u_6$
	u_0	9·0000						
0	x_1	1·0000	0·0000					
1	x_2	-0·5000	0·0000	-0·2500 ←				
	x_3	3·0000	0·0000	1·5000	-9·0000			
	x_4	0·5000	0·0000	-0·2500	1·5000	-0·2500		
upper bounds								
1	u_5	3·0000	-1·0000	-0·5000	8·0000	1·5000	-3·0000	
1000	x_6	999·5000	0·0000	0·2500	-1·5000	0·2500	-0·5000	-0·2500

Tableau 12 ≡ Node 4

lower bounds		x_0	$-u_1$	$-x_2$	$-u_3$	$-u_4$	$-x_5$	$-u_6$
	u_0	8·0000						
0	x_1	1·0000	0·0000					
1	u_2	2·0000	0·0000	-4·0000				
	x_3	0·0000	0·0000	6·0000	0·0000			
	x_4	1·0000	0·0000	-1·0000	0·0000	0·0000		
upper bounds								
1	u_5	4·0000	-1·0000	-2·0000	5·0000	2·0000	-4·0000	
1000	x_6	999·0000	0·0000	1·0000	0·0000	0·0000	0·0000	0·0000

bounds and to illustrate the branching process. $[x_1 = 1, x_2 = 0]$, Node 3 and $[x_1 = 1, x_2 = 1]$, Node 4 represent two equally good integer solutions. The tree of subproblems is exhausted after developing node 4. Hence these two solutions constitute two alternative optima.

d. Statement of the algorithm

The steps leading to the solution of the worked example in the last section can be formalized and stated as a complete algorithm. However,

first a description of the store layout is given. Let STACK denote the area of store where the tree of subproblems can be stored. Let S_i denote an intermediate area of store where an intermediate problem can be stored. Let RP denote a rule for choosing a problem out of STACK, and let RV denote a rule for choosing the index of a variable which is required to be integral but which is not integral in the problem being currently investigated. The steps of the algorithm are stated below.

Step 1 Given the problem Q, ignore the integer constraints (7.4) and apply the primal algorithm to the problem $\bar{Q}$; at the end of this step the tableau is dual feasible.

Step 2 On the dual feasible tableau of the problem $\bar{Q}$, apply the dual algorithm. If the current $\bar{Q}$ problem is infeasible then goto logical step 3 i.e. delete this node. At the end of this step the tableau provides the optimal solution to the problem $\bar{Q}$.

Logical Step 0 If it is the optimal solution of the first $\bar{Q}$ problem i.e. the problem at the stem, then goto step 3.

Logical Step 1 If the cost of the optimal solution to the current $\bar{Q}$ problem is less than Z_Q, the cost of the currently best feasible solution, then goto logical step 3, i.e. this node is deleted.

Logical Step 2 If the solution of the current convex $\bar{Q}$ problem is a feasible solution i.e. (7.4) is satisfied, then goto step 8.

Step 3 Store the current problem in STACK.

Logical Step 3 If S_i is full then transfer S_i to the main tableau area and goto step 2; S_i is empty at the end of this step.

Logical Step 4 If STACK is empty then goto Exit.

Step 4 Choose a problem from the STACK applying the rule RP.

Step 5 Choose a variable index i from the problem tableau by the application of the rule RV and produce two convex problems $\bar{Q}(ub)$ and $\bar{Q}(lb)$.

Step 6 Store $\bar{Q}(lb)$ in S_i; S_i is now full.

Step 7 With $\bar{Q}(ub)$ in the main problem area of the store goto step 2.

Step 8 Update Z_Q i.e. overwrite the best solution so far by the currently best feasible solution and goto logical step 3.

Exit: If no feasible solution has been obtained so far then Q has no feasible solution, otherwise the currently best feasible solution is the optimal solution.

e. Computer implementation of the algorithm and discussions

In all, three branch and bound algorithms have been developed and coded, these are described in chapters 7, 8, 9 of the Part 3 of the thesis. The layout of the backing store and the mechanics of transferring problems to and from the backing store and the main tableau area are essentially the same, and since these are described in detail in Chapter 9, the same description is not repeated here. However, other pertinent details of the program are discussed in what follows. The primal/dual quadratic programming code outlined in Chapter 5 constitutes the main core of the algorithm considered in this chapter and described in Section d. However, to help understand the Algol program given in Appendix 10 the important identifiers and procedures are best elaborated in the form of Table 7.1.

Identifier or Procedure	Type	Description
int	integer	Defines the set I, such that the variables with indices 1,2,...int are required to be integer.
length	integer	Defines the area of the store in terms of the largest index of a one dimensional array, in which all the information regarding a subproblem can be stored.
send	procedure	sends a subproblem with additional information, a[1:length] to the backing store.
fetch	procedure	Fetches a problem a[1:lengh] from the backing store.
intst	bøølean	=true if the tableau is in integer phase =false if the tableau is in continuous phase } of optimization
toler	real	The accuracy within which each variable is taken to be integer.
cbest[0:n+m+int]	real array	Locations at which the current best feasible solution is stored.
opt	boolean	=true if a feasible integer solution is currently available =false otherwise.
Store[1:ub] lim avlist[0:lim]	real array integer integer array	} Together these define a stack mechanism, which is fully described in Chapter 9

Table 7.1

The rule RP of choosing a problem from the stack for investigation can have many variations, however, for simplicity a last-in-first-out scheme has been coded. The rule R.V for selecting a non-integral variable to branch on can similarly incorporate many alternative heuristics. For

integer linear programs, Beale and Small [3.4] have adopted Little et al's [3.2] heuristics, for choosing that variable to branch on which makes one of the branches as bad as possible; they then investigate the problem in the other branch. However, with quadratic objective functions the evaluation of the expected change in its value by branching on a variable is rather complicated, hence a random rule has been incorporated i.e. the index i is chosen at random such that $x_i \neq 0 \bmod(1)$ and $i \in I$. Ten of the thirty problems solved by the technique described in chapter 4 were also solved by the present code. These ten problems were solved both with linear and quadratic objective functions. With linear problems the use of the enlarged quadratic programming tableau and the use of the time consuming double pivot operation made the figures for computer time unattractive. However, in spite of the very simple strategies adopted for RP and RV the results in terms of the number of simplex iterations compared favourably with those of the figures reported by Dr Richards [2.28], and often fared better. It was intended to perform a large scale experiment by incorporating variants for RP and RV and then pick out the best heuristics on the basis of these tests. However, while investigating a class of integer programming problems, viz, the fixed charge problems, a more attractive method was developed. The experiments with branching heuristics therefore were conducted for these type of problems and are reported in Chapter 9. Ten of the fixed charges problems discussed in Chapter 9 were solved by this method. Discussion of the advantage of solving these problems by this improved method is deferred until the end of Chapter 9, where the relevant figures regarding the problem dimension and computing time are compared.

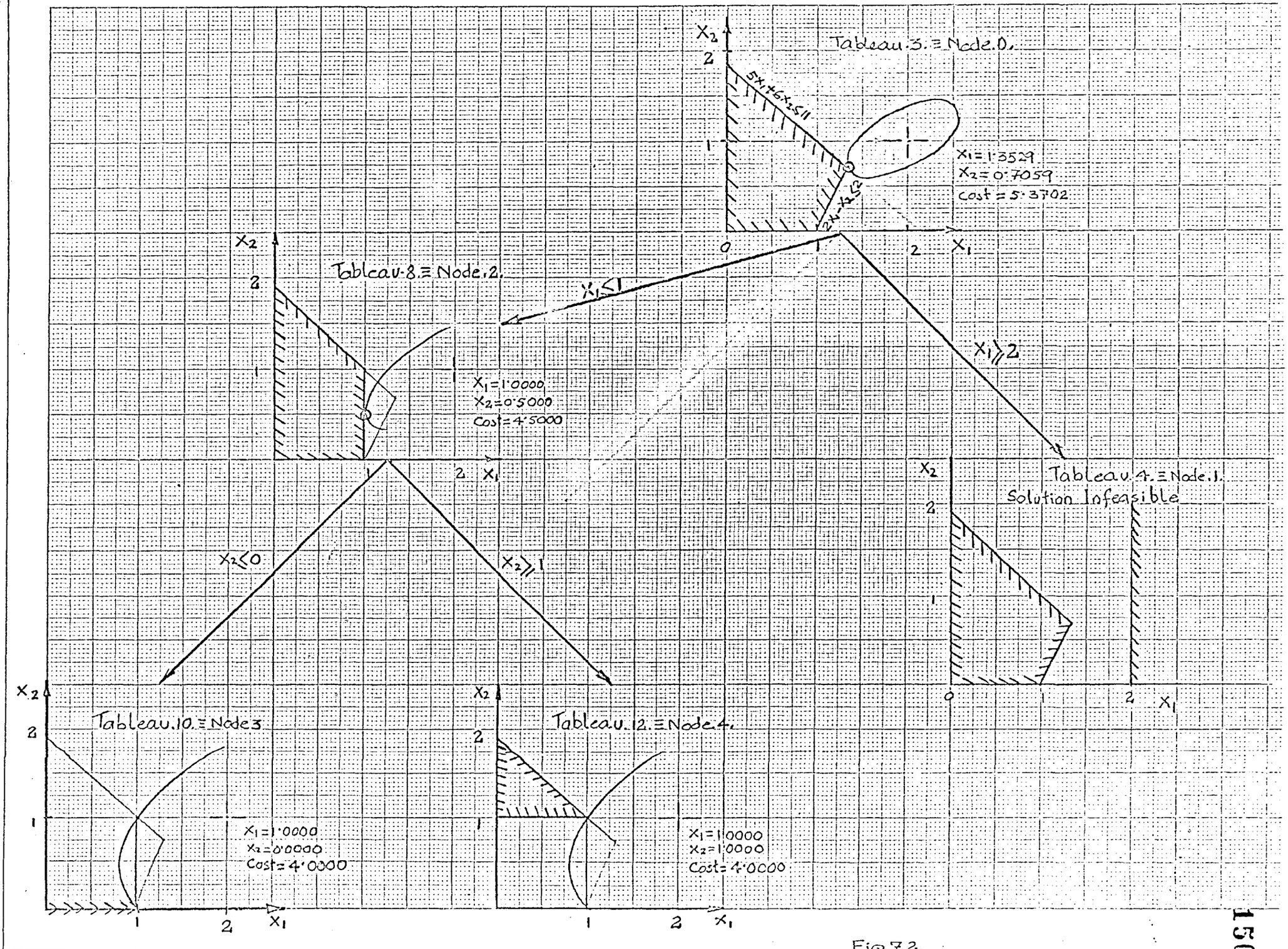


Tableau.3. ≡ Node.0.
$5x_1+6x_2 \leq 11$
$2x_1-x_2 \leq 2$
$x_1 = 1.3529$
$x_2 = 0.7059$
Cost = 5.3702
$x_1 \leq 1$
$x_1 \geq 2$
Tableau.8. ≡ Node.2.
$x_1 = 1.0000$
$x_2 = 0.5000$
Cost = 4.5000
Tableau.4. ≡ Node.1.
Solution Infeasible
$x_2 \leq 0$
$x_2 \geq 1$
Tableau.10. ≡ Node 3
$x_1 = 1.0000$
$x_2 = 0.0000$
Cost = 4.0000
Tableau.12. ≡ Node.4.
$x_1 = 1.0000$
$x_2 = 1.0000$
Cost = 4.0000
x_1
x_2

Fig.7.2.

Chapter 6

Dichotomizing Procedure For The Fixed Charge Transportation Problem

Introduction

An important problem in the area of realistic transportation problems is the fixed charge problem where a fixed charge is considered in addition to the variable cost. It has been discussed by Kuhn and Baumol [3·8] and they point out the importance of solving such problems. As solution procedures they recommend parametric methods of investigating the vertices adjacent to the optimal solution of the associated linear transportation problem and forcing degeneracy where possible. In an earlier paper Balinski [3·9] developed a powerful method of obtaining a good approximate solution and a lower bound to the optimal solution of this problem. Dwyer [3·10] has recently discussed the problem again. He suggests applying reduced matrices (reduced by the parametric equations of the reduced cost coefficients) with the objective of forcing degeneracy in the optimal tableau of the corresponding linear transportation problem. However, neither Balinski's method nor, for unequal fixed charges, Dwyer's method or that of Kuhn and Baumol provides the true optimum. Moreover, contrary to the claims made Dwyer's method does not appear to provide the optimal assignment even when the fixed charges are equal, a point discussed in Section f.

In Section .a of this chapter, the problem is stated and a method of obtaining an approximate solution is considered. Section .b, discusses a branching process, while Section .c proves the finiteness of the search leading to the global optimum. Section .d provides a worked example. The algorithm is formally stated in Section .e. In

Section .f, the solution of a representative example from [3·9] and some points concerning the computer implementation of the method are discussed.

a. Statement of the problem and an approximate solution

Define a problem P:

$$\text{Minimize } Z(X,Y) = \sum_{i=1}^{m} \sum_{j=1}^{n} (c_{ij}x_{ij} + d_{ij}y_{ij}) \qquad (8\cdot1)$$

subject to the linear constraints

$$\left.\begin{aligned} \sum_{j=1}^{n} x_{ij} &= a_i \ , \ i=1,2,\ldots m \\ \sum_{i=1}^{m} x_{ij} &= b_j \ , \ j=1,2,\ldots n \end{aligned}\right\} \quad x_{ij} \geq 0 \text{ for all } i,j, \qquad (8\cdot2)$$

and the non-linear constraints

$$\left.\begin{aligned} y_{ij} &= 1 \quad \text{if} \quad x_{ij} > 0, \\ y_{ij} &= 0 \quad \text{if} \quad x_{ij} = 0 \end{aligned}\right\} , \qquad (8\cdot3)$$

where a_i, b_j, c_{ij}, d_{ij} are non-negative integers for all i,j and $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$.

If constraint (8·3) is replaced by

$$0 \leq y_{ij} \leq 1 \ , \ y_{ij} \text{ integer, and,} \qquad (8\cdot4)$$

$$0 \leq x_{ij} \leq m_{ij}y_{ij} \ , \ m_{ij} = \min (a_i, b_j), \qquad (8\cdot5)$$

the problem can be regarded as one of integer programming. However, application of an integer programming algroithm cannot be considered because

(a) it does not take advantage of the unitary matrix of the constraint equations (8·2), and

(b) integer programming algorithms to date do not tackle efficiently problems of such dimensions as may be expected for realistic values of m and n.

The formulation given above is identical to that of Balinski [3·9] and the further development of the next section is motivated by the theory given in his paper. The most pertinent deduction made in [3·9] can be stated as follows. Let $\bar{P}$ denote the problem P with the integer constraints on the y_{ij} variables ignored. In the optimal solution to the problem $\bar{P}$ the inequalities (8·5) are satisfied as exact equalities [3·9], i.e., $x_{ij} = m_{ij}y_{ij}$ for all i,j. If one defines a problem Q:

$$\text{Minimize } Z_Q(X) = \sum_{i=1}^{m} \sum_{j=1}^{n} \left(c_{ij} + \frac{d_{ij}}{m_{ij}}\right) x_{ij} \qquad (8·6)$$

Subject to (8·2),

then $Z_Q(X^o)$ the cost of the optimal solution to the problem Q is the same as $Z_{\bar{P}}$ the cost of the optimal solution to the problem $\bar{P}$. The solution of the problem Q, (8·6) therefore leads to the following bounds

$$Z_Q(X^o) \leq Zopt \leq Z_P(X^o, Y^o) \qquad (8·7)$$

where

Zopt denotes the cost of the global optimum solution to the problem P, $Z_P(X^o, Y^o)$ the cost of the solution (X^o, Y^o) to P, (X^o, Y^o) being trivially obtained using (8·3).

b. A branching process and improving the lower bound

In the last section it was noted that the application of an integer programming algorithm is not worthwhile, however, one could take advantage of the structure of the problem and apply 0-1 variable technique as follows. Consider the constraints (8·4) of the problem P. If one branched on any variable y_{st} of this problem, then one could define two subproblems,

(a) $P(y_{st}=1)$ which is the same as problem P with the additional constraint $y_{st} = 1$, and

(b) $P(y_{st}= 0)$ which is the same as problem P with the additional constraint $y_{st} = 0$.

Let Zopt $P(y_{st}= 1)$ denote the cost of the optimal solution to the problem $P(y_{st}= 1)$ and, Zopt $P(y_{st}= 0)$ denote the cost of the optimal solution to the problem $P(y_{st}= 0)$. The union of the solution spaces of the problems $P(y_{st}= 1)$ and $P(y_{st}= 0)$ produces the solution space of the problem P. Therefore, it must be true that,

$$\text{Zopt} = \min \{\text{Zopt } P(y_{st}= 1), \text{ Zopt } P(y_{st}= 0) \}. \qquad (8\cdot8)$$

The approximation of Balinski (Section a) hinges on the condition that in the optimal solution to the problem $\bar{P}$, $x_{ij}=m_{ij}y_{ij}$. If $\bar{P}(y_{st}= 1)$ denotes the problem corresponding to the subproblem $P(y_{st}= 1)$ with the integer constraints on the y_{ij} variables ignored, then it can be shown as in [3·9] that in the optimal solution of $\bar{P}(y_{st}= 1)$, $x_{ij}= m_{ij}y_{ij}$ for all i,j (excepting i=s and j=t of course). Further, the constraint $0\leq x_{st}\leq m_{st}y_{st}$ must be satisfied since $y_{st}= 1$, and constraints (8·2) ensure that $x_{st} \leq m_{st}$. If one solves the problem $Q(y_{st}= 1)$ defined as follows

$$\text{Minimize } Z_{Q(y_{st}= 1)}(X)= \sum_{i=1}^{m} \sum_{j=1}^{n} \Pi_{ij}x_{ij} + d_{st}y_{st} \qquad (8\cdot9)$$

subject to the constraints (8·2) and $y_{st}= 1$, where $\Pi_{ij}= c_{ij} + \frac{d_{ij}}{m_{ij}}$ of the problem Q for all i,j except i=s and j=t and $\Pi_{st}= c_{st}$, then the cost of the optimal solution to this problem $Z_{Q(y_{st}= 1)}$ will be the same as the cost $Z_{\bar{P}(y_{st}= 1)}$ of the optimal solution to the problem $\bar{P}(y_{st}= 1)$. Let $Z_{Q}'(y_{st}= 1)$ denote the cost of the optimal solution to the problem $Q'(y_{st}= 1)$:

$$\text{Minimize } Z_{Q}'(y_{st}= 1)(X) = \sum_{i=1}^{m} \sum_{j=1}^{n} \Pi_{ij}x_{ij} \qquad (8\cdot10)$$

subject to (8·2) where $\Pi_{ij} = \Pi_{ij}$ of (8·9).

Then it is obvious that $Z_{Q(y_{st}=1)} = Z_{Q'(y_{st}=1)} + d_{st}$.

This establishes the following lower bound on Zopt $P(y_{st}=1)$.

$$Z_{Q'(y_{st}=1)} + d_{st} = Z_{Q(y_{st}=1)} = Z_{\bar{P}(y_{st}=1)} \leq \text{Zopt } P(y_{st}=1) \quad (8\cdot11)$$

Similarly, let $\bar{P}(y_{st}=0)$ denote the problem corresponding to the problem $P(y_{st}=0)$ with the integer constraint on rest of the variables y_{ij} ignored. It can again be shown that in the optimum solution of $\bar{P}(y_{st}=0)$, $x_{ij} = m_{ij}y_{ij}$ for all i,j, and $x_{st} = y_{st} = 0$ since $x_{st} \leq m_{st}y_{st}$ needs to be satisfied. If one solves the problem, $Q(y_{st}=0)$ as defined below

$$\text{Minimize } Z_{Q(y_{st}=0)}(X) = \sum_{i=1}^{m}\sum_{j=1}^{n} \Pi_{ij}x_{ij} \quad (8\cdot12)$$

subject to (8·2) and $y_{st}=0$, $x_{st}=0$, where $\Pi_{ij} = c_{ij} + \frac{d_{ij}}{m_{ij}}$

for all i,j, then the cost of the optimal solution to (8·12) $Z_{Q(y_{st}=0)}$ will be the same as the cost of the optimal solution to $\bar{P}(Y_{st}=0)$. Consider the problem $Q'(y_{st}=0)$

$$\text{Minimize } Z_{Q'(y_{st}=0)}(X) = \sum_{i=1}^{m}\sum_{j=1}^{n} \Pi_{ij}x_{ij} \quad (8\cdot13)$$

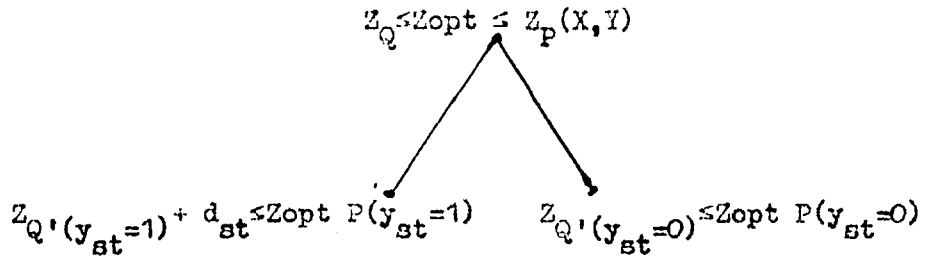
subject to (8·2) where $\Pi_{ij} = \Pi_{ij}$ of (8·12) for all i,j and $\Pi_{st} = M$ a very large cost coefficient. If $Z_{Q'(y_{st}=0)}$ denotes the cost of the optimal solution of this problem then it can be shown trivially that $Z_{Q'(y_{st}=0)} = Z_{Q(y_{st}=0)}$.

The following bound is therefore established

$$Z_{Q'(y_{st}=0)} = Z_{Q(y_{st}=0)} = Z_{\bar{P}(y_{st}=0)} \leq \text{Zopt } P(y_{st}=0). \quad (8\cdot14)$$

It is of interest to note that (8·11) and (8·14) can be deduced by the principle of optimality type argument of dynamic programming.

The relations (8·7), (8·8), (8·11), (8·14) are illustrated in the two branches of Fig. 8·1.



$$Zopt = \min \{Zopt\ P(y_{st}=1),\ Zopt\ P(y_{st}=0)\}$$

Fig. 8·1

So far, starting from the problem P, only a single step in the branching process has been discussed, but these arguments can be extended in general to any depth of branching. Further, the branching can be started on any of the variables y_{ij} of the problem P. However, it is advantageous to start from the optimal solution X^o of the problem Q and choose a variable $x^o_{st}>0$ and branch on its corresponding fixed charge variable y_{st}. It will be shown in the next section that $Z_{Q'}(y_{st}=1) + d_{st} \geq Z_Q$ and $Z_{Q'}(y_{st}=0) \geq Z_Q$, i.e., whenever a branching is undertaken the lower bounds are improved or at worst held at the same value. The lower bounds shown at each node of the bifurcating tree which is being so defined are local to the branches emanating from it. However, the currently available minimum cost feasible solution is the global upper bound. The search, therefore, consists in defining new subproblems of type (8·10) and (8·13), solving for better lower bounds (8·11), (8·14) and the corresponding feasible solutions to the problem P are checked for better upper bounds. The search is terminated when all the local lower bounds at the developed nodes become equal to or greater than the current best upper bound.

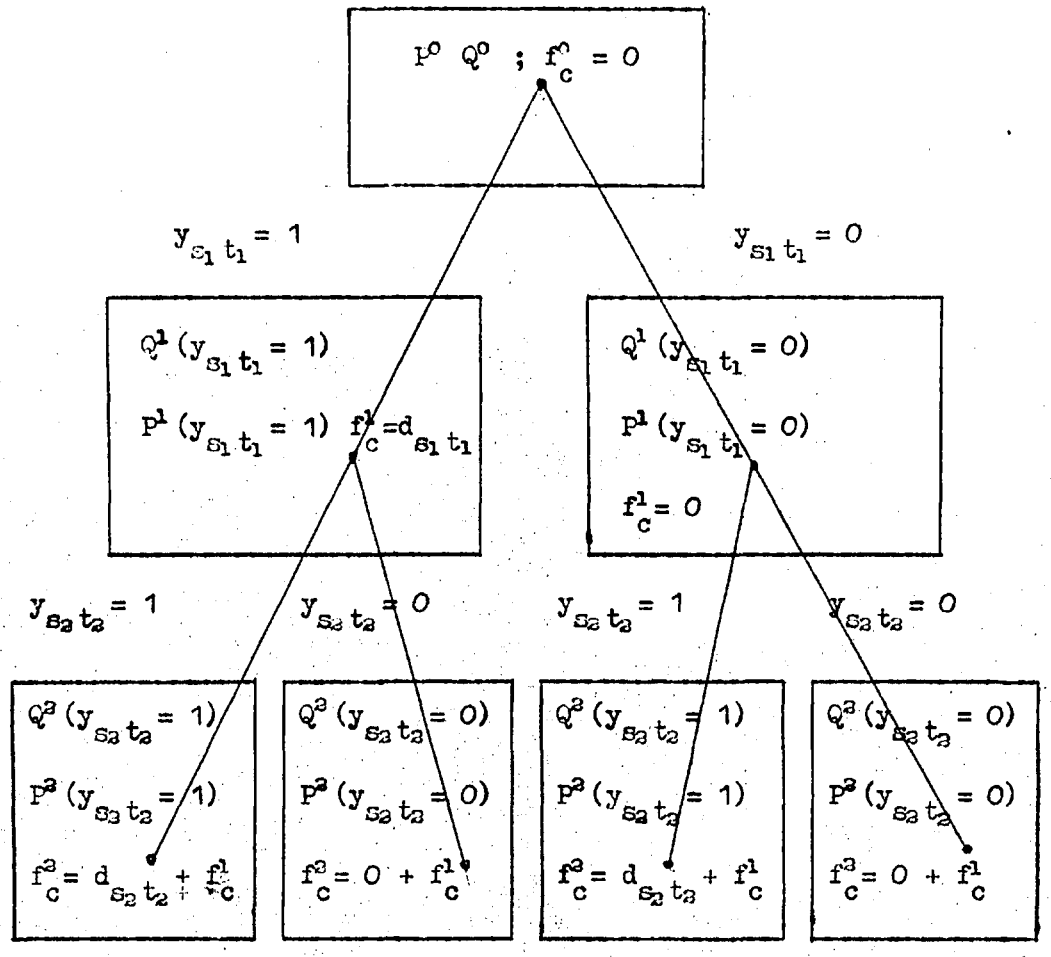
c. Behaviour of the lower bound and finiteness

Let the original problems P and Q be relabelled P^0 and Q^0. The variables shown in Fig. 8·2, illustrating further details of the branching process are explained below.

Let

y_{siti}	denote the fixed charge variable which is being dichotomized at the current level i,
f_c^i	denote the cumulative fixed charges at the current node corresponding to y_{siti} which are set equal to 1, (the relation by which f_c^i is updated at each node is given later (8.21)),
$P^i(y_{siti}= 1)$ $P^i(y_{siti}= 0)$	represent the two subproblems of type P emerging from the predecessor problem P^{i-1},
$Q^i(y_{siti}= 1)$ $Q^i(y_{siti}= 0)$	represent the two derived subproblems of type Q, which may be considered emerging from the predecessor problem Q^{i-1}.

Q' problems (8·10) and (8·13) are the actual problems solved and are conceptually same as the corresponding Q Problems. In the above notation the index i denotes the depth of any node of the tree, illustrated in Fig. 8·2. To avoid cumbersome notation (s,t), will not be subscripted in the subsequent derivation. Let the



$$\left.\begin{aligned} Z_Q{}^i(y_{s_i t_i} = 0) &= Z_{Q'}{}^i(y_{s_i t_i} = 0) + f_c^i \\ Z_Q{}^i(y_{s_i t_i} = 1) &= Z_{Q'}{}^i(y_{s_i t_i} = 1) + f_c^i \end{aligned}\right\}$$

and $Z_Q{}^i(y_{s_i t_i} = 0) \leq \text{Zopt } P^i(y_{s_i t_i} = 0)$ and $Z_Q{}^i(y_{s_i t_i} = 1) \leq \text{Zopt } P^i(y_{s_i t_i} = 1)$

$i = 0,1,2 \ldots nxm.$

In general the above relations hold at any node of the tree.

Fig. 8·2.

distinction between $y_{st} = 0$ and $y_{st} = 1$ type problems be dropped and Z_{Qi} denote the cost of the optimal solution to the problem Q^i and X^i the corresponding optimal assignment. Let X^{i+1} denote the assignment of the problem $Q'^{\,i+1}(y_{st} = 1)$, (8·10) and $Z_{Q\,i+1}(y_{st} = 1)$ the cost of the

optimal solution. The cost of these two optimal solutions are given by

$$Z_{Qi} = x^i_{st}\left(\frac{d_{st}}{m_{st}} + c_{st}\right) + C^i \qquad (8\cdot15)$$

and

$$Z_{Q'\,i+1}(y_{st}=1) = x^{i+1}_{st} \cdot C_{st} + C^{i+1} \qquad (8\cdot16)$$

where C^i denotes the cost of the rest of the assignment X^i excluding $x^i_{st} > 0$, and C^{i+1} denotes the cost of the rest of the assignment X^{i+1} excluding the contribution of x^{i+1}_{st}. Now X^{i+1} is a feasible assignment of the problem Q^i but X^i is the optimal assignment, hence,

$$x^{i+1}_{st}\left(\frac{d_{st}}{m_{st}} + C_{st}\right) + C^{i+1} - x^i_{st}\left(\frac{d_{st}}{m_{st}} + C_{st}\right) - C^i \geq 0 \qquad (8\cdot17)$$

By (8·15) and (8·16)

$$Z_{Q'\,i+1}(y_{st}=1) - Z_{Qi} + x^{i+1}_{st}\frac{d_{st}}{m_{st}} \geq 0$$

or

$$Z_{Q'\,i+1}(y_{st}=1) - Z_{Qi} + d_{st} \geq \left(\frac{m_{st} - x^{i+1}_{st}}{m_{st}}\right) d_{st}$$

For $m_{st} \geq x_{st}$ and $d_{st} \geq 0$ the above reduces to

$$\left(Z_{Q'\,i+1}(y_{st}=1) + d_{st}\right) - Z_{Qi} \geq 0. \qquad (8\cdot18)$$

Hence, the lower bound to the cost of the optimal solution of the new subset of P problems i.e. the cost of the optimal solution to Q^{i+1} problem, $Z_{Qi+1}(y_{st}=1) = \left(Z_{Q'\,i+1}(y_{st}=1) + d_{st}\right)$ is greater than or equal to the cost Z_{Qi} of the optimal solution to the Q^i problem, at the predecessor node. Consider now the problem $Q'^{\,i+1}(y_{st}=0)$ (8·13) and let X^{i+1} denote the optimal assignment and $Z_{Q'\,i+1}(y_{st}=0)$ the cost of the optimal solution. Then

$$Z_{Q'\,i+1}(y_{st}=0) = Mx^{i+1}_{st} + C^{i+1} \qquad (8\cdot19)$$

where C^{i+1} is the cost of the rest of the assignment X^{i+1} excluding x^{i+1}_{st}. The high penalty cost M ensures $x^{i+1}_{st} = 0$, however, if this is not true then it represents an infeasibility in the corresponding Q

and P problems (8·12) which require $x_{st}^{i+1} = 0$ and this branch will be priced out. If $x_{st}^{i+1} = 0$ then $Z_{Qi+1}(y_{st} = 0) = Z_{Q',i+1}(y_{st} = 0) = C^{i+1}$ the cost of a feasible assignment of the problem Q^i since the constraint set of Q^i is a subset of the constraints Q'^{i+1} etc. Hence,

$$Z_{Q',\,i+1}(y_{st} = 0) - Z_{Qi} \geq 0 \qquad (8\cdot20)$$

The quantity f_c^{i+1} is also updated in the two branches as follows,

$$f_c^{i+1} = f_c^i + d_{st} \text{ in the branch } y_{st} = 1 \qquad (8\cdot21)$$

$$f_c^{i+1} = f_c^i \text{ in the branch } y_{st} = 0.$$

The general expression for the lower bounds on both the branches can be written as $Z_Q^{i+1} = Z_{Q',\,i+1} + f_c^{i+1}$, and the lower bounds have been shown to be (8·18), (8·20), improvements on Z_{Qi}.

Thus there is a potential increase in the lower bounds down the branches of the tree. To prove that the algorithm terminates in finite number of steps one needs to show that the growth of the tree is finite. At worst, it may be necessary to branch on all the mxn variables y_{ij}. In this case the total no. of nodes developed will be $\sum_{i=0}^{mxn} 2^i = 2^{mxn+1} - 1$ which is a finite number. Although this is a frighteningly large number, in practical problems because of the facility of pruning the tree whenever the global upper bound is exceeded by the local lower bound at a node, the tree developed is of much reduced size. Though the proof appears rather tortuous an intuitive understanding can easily be gained by going through the worked example of a dichotomy in the next section.

d. Example

Consider the fixed charge transportation problem P (taken from [3·9] and modified) given by the data:

c_{ij}, d_{ij}:

			a_i:
2,10	3,30	4,20	10
3,10	2,30	1,25	30
1,10	4,30	3,20	40
4,10	5,30	2,20	20
b_j: 20	50	30	

(8·22)

The derived problem Q is given by

$(c_{ij} + \frac{d_{ij}}{m_{ij}})$:

$m_{ij} = \min(a_i, b_j)$

3·00	6·00	6·00
3·50	3·00	1·83
1·50	4·75	3·67
4·50	6·50	3·00

(8·23)

The solution of this derived transportation problem is shown in (8·24).

x^o_{ij}:

0	10	0
0	20	10
20	20	0
0	0	20

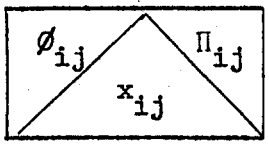
$Z_Q(X^o) = 323·3$ (8·24)

The corresponding $Z_p = 365$ the cost of the fixed-charge problem P is obtained by (8·24), (8·22) and relation (8·3).

Hence the bounds to the optimal solution are given by

$$323·3 \leq Z_{opt} \leq 365.$$

Introducing dual variables U_i and V_j and the reduced costs $\emptyset_{ij} = \Pi_{ij} - U_i - V_j$, the optimal tableau of the problem Q is given by



u_i \ v_j	0	3·25	2·08
2·75	0·25 / 3·00 / 0	0 / 6·00 / 10	1·17 / 6·00 / 0
-0·25	3·75 / 3·50 / 0	0 / 3·00 / 20	0 / 1·83 / 10
1·5	0 / 1·50 / 20	0 / 4·75 / 20	0·09 / 3·67 / 0
0·92	3·58 / 4·50 / 0	3·33 / 6·50 / 0	0 / 3·00 / 20

(8·25)

Branching on the variable y_{23} ($x_{23} > 0$) the two immediate subproblems can be written in terms of the cost matrices.

$Q^{11}(y_{23} = 1)$:

3·00	6·00	6·00
3·50	3·00	1·00 *
1·50	4·75	3·67
4·50	6·50	3·00

$f_c^1 = 25$ (8·26)

$Q^{12}(y_{23} = 0)$

3·00	6·00	6·00
3·50	3·00	999 *
1·50	4·75	3·67
4·50	6·50	3·00

$f_c^1 = 0$ (8·27)

M=999 a large cost coefficient

To solve (8·26) and (8·27) one does not need to solve again two new problems. In what follows a method (a parametric method) will be described to solve the subproblems from the assignment of the predecessor problem (8·25).

U_i \ V_j	0	3·25	1·25
2·75	0·25 / 3·00 / 0	0 / 6·00 / 10	2·00 / 6·00 / 0
-0·25	3·75 / 3·50 / 0	0 / 3·00 / 20	0 / 1·00 / 10
1·5	0 / 1·50 / 20	0 / 4·75 / 20	0·92 / 3·67 / 0
1·75	2·75 / 4·50 / 0	1·50 / 6·50 / 0	0 / 3·00 / 20

$f_c^1 = d_{33} = 25$

$Z_{Q'^1}(y_{33}=1) = 315$

(8·28)

Substitute the assignment of (8·25) in (8·26) and proceed to compute new U_i and V_j^s from the m+n-1 equations $U_i + V_j = \Pi_{ij}$, for $x_{ij} > 0$.

One can now proceed by a primal method. However, in this case the tableau (8·28) is still optimal. It is to be noted that $Z_{Q'^1}(y_{33}=1) + f_c^1 = 340$ is now a better lower bound on the subset of P problems in this branch. Treating, Q'^1 $(y_{33}=0)$ similarly, one obtains the tableau (8·29).

U_i \ V_j	0	3·25	999·25
2·75	0·25 / 3·00 / 0	0 / 6·00 / 10	-996 / 6·00 / 0
-0·25	3·75 / 3·50 / 0	0 / 3·00 / 20	0 / 999 / 10
1·5	0 / 1·50 / 20	0 / 4·75 / 20	-997·08 / 3·67 / 0
-996·25	1000·75 / 6·50 / 0	999·78 / 6·50 / 0	0 / 3·00 / 20

(8·29)

Tableau (8·29) is not optimal.

Forming a μ-loop with the cells of x_{23}, x_{22}, x_{32}, x_{33} one obtains the optimal assignment of tableau (8·30).

U_i \ V_j	0	3·25	2·17
2·75	·25 (0) 3·00	0 (10) 6·00	1·08 (0) 6·00
-0·25	3·75 (0) 3·50	0 (30) 3·00	997·8 (0) 999·0
1·5	0 (20) 1·50	0 (10) 4·75	0 (10) 3·67

$f_c^1 = 0$ (8·30)

$Z_{Q'^1}(Y_{23}=0) = 324·2$

$Z_{Q'^1}(Y_{23}=0) + f_c^1 = 324·2$ is again a better lower bound on the subset of P problems in this branch. The corresponding actual cost of the new assignment (X^1, Y^1) is found to be an improved solution (upper bound), $Z_p(X^1, Y^1) = 360$.

Once a better upper bound is found it becomes a global upper bound. However, the lower bounds are local to the branches emanating from the current node. The situation at the end of depth one of dichotomy is illustrated in Fig. 8·3.

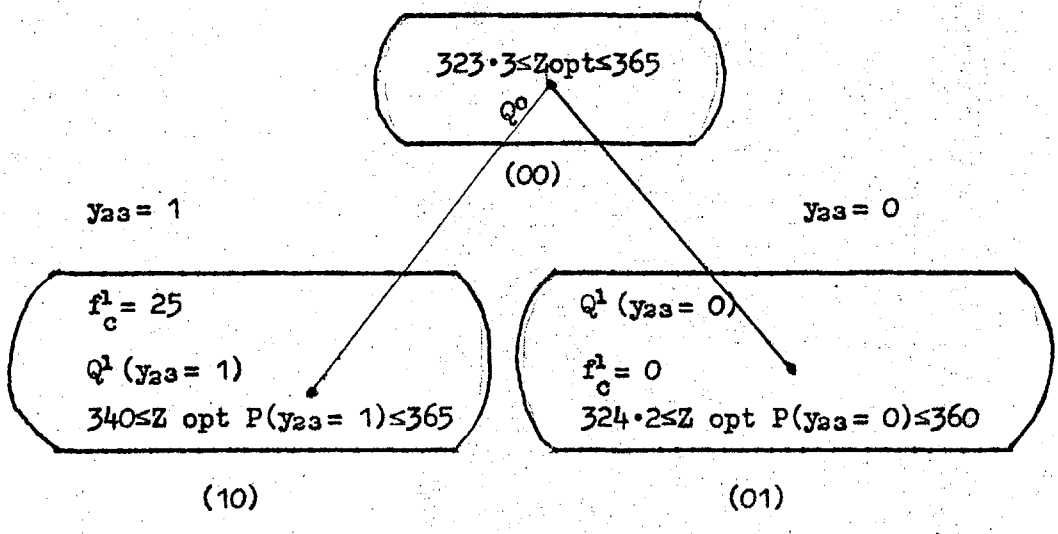


Fig. 8·3

When node (01) is developed the bounds at node (10) read $340 \leq ZoptP(y_{3a}=1) \leq 360$. Zopt=360, is the cost of the optimal solution to the problem; proceeding in the above fashion after developing 35 nodes this optimality is established.

If a primal-dual flow algorithm is being used for the solution of the transportation problem, the following procedure can be suggested.

Subproblem $Q'(y_{st}=1)$: Delete the labelled zeros and hence the corresponding assignments of the row s or column t (or ideally the sum of whichever is minimum). Adjust U_s or V_t accordingly, so as to retain the zero label at Π_{st}. Solve the new restricted primal by the flow method and follow the standard algorithm.

Subproblem $Q'(y_{st}=0)$: Delete the zero label and the current assignment at $\Pi_{st}=M$. Solve the restricted primal and follow the standard algorithm.

e. Formal statement of the algorithm

Before formally stating the algorithm a description of the store layout is given. Let STACK denote a backing store that will hold any number of subproblems representing the still undeveloped nodes of the tree. There can be various heuristics of choosing problems out of this STACK. For simplicity let it be assumed that last in first our scheme is employed. Let S_i denote an area of the store where intermediate problems can be stored. Further, let a rule R1 be defined to choose the index pair (s,t) of the variable to branch on.

R1: $(s,t) \in S1$ and <u>not</u> S2 (8·31)

where S2 denotes the set of index pairs of the y variables already dichotomized, and S1 denotes the set of index pairs of the x variables in the optimal basic solution of the Q problem at the current node.

It can be shown that at any of the feasible nodes the above set will not be empty.

The steps of the algorithm are stated below.

Step 1 Given the problem P derive problem Q.

Step 2 Solve the problem Q to obtain Z_Q and Z_P.

Step 3 By rule R1 choose (s,t) the index pair of a variable to branch on and produce two subproblems $Q'^{i+1}(y_{st}=1)$ and $Q'^{i+1}(y_{st}=0)$ from the problem Q'^{i}.

Step 4 Store $Q'^{i+1}(y_{st}=0)$ in S_i and goto step 6.

Step 5 Bring back $Q'^{i+1}(y_{st}=0)$; S_i is now empty.

Step 6 From the assignment of Q'^{i} solve Q'^{i+1} by parametric method.

Logical Step 1 If the cost of the corresponding solution to the P problem is cheaper than the best solution obtained so far then overwrite it with the current solution.

Logical Step 2 If the new local lower bound is greater than the cost of the current best solution then goto logical step 3 (i.e. this branch will be deleted).

Step 7 Store the problem matrix and other information in STACK.

Logical Step 3 If S_i is full then goto step 5.

Logical Step 4 If STACK is empty then goto Exit.

Step 8 Pick a problem from the STACK and goto step 3.

Exit: Print the current best solution which is the optimal solution.

f. Discussions

Naturally the real-life problem supplied in [3·9] was investigated and to the surprise of the author a very much improved assignment was produced. The first approximate solution produces the bounds $451{\cdot}05 \leq Z_{opt} \leq 504{\cdot}53$.

The author's solution which is of the highest order of degeneracy is given below. There is only one entry per column. The cost of this optimal assignment is 294·55 + 177= 471·55 where

												a_i:
	15											15
		20										20
20				5	20							45
						30					5	35 (8·32
			15							10		25
								35				35
							10					10
									25			25
b_j:	20	15	20	15	5	20	30	10	35	25	10	5

the first term is the sum of the variable costs and second term the sum of the fixed charges.

It does not seem to be clear whether by Dwyer's method [3·10] one can force degeneracy in a non-optimal but feasible basis (present case). If this can be done then it appears optimal assignment with equal fixed charges can also be obtained. However, if fixed charges are set at an arbitrarily low value say ϵ then the 17 route soulution given by Balinski Z= 266·70 + 17 ϵ will be the optimal solution and not the current one Z= 294·55 + 12 ϵ. Hence, the method outlined in [3·10] does not guarantee optimality even with equal fixed charges. This is to be logically expected, since the relationship between θ_{ij} and d_{ij}

is not investigated in this method as opposed to Balinski's method [3·9].

The solution given in (8·32) was produced after developing 1008 nodes i.e., solving as many subproblems. Numerous intermediate assignments in the range $471·55 \leq Z_{opt} \leq 504·5$ were also found in the process. The rule R1 mentioned in the previous section, and the rule for choosing the next problem from the stack are very important, together, they determine uniquely the growth of the tree.

The program used for the purpose of experimentation was written in Algol and is to be found in Appendix II. The procedure scale, scales the cost coefficients of the derived transportation problem to integers. Procedure fetch and send arrange the transfer of the problems to and from the tree. The core of the program uses a library procedure for the solution of the transporation problem [3·15]; unfortunately, this needed to be debugged. Further, this procedure solves the transportation problem by the primal-dual algorithm, hence the otherwise elegant parametric approach of the primal method could not be implemented; the procedure also destroys the cost coefficients of the original tableau if called by name. The branching scheme was therefore implemented in a simple-minded and inefficient fashion of solving each problem at a node all over again. However, at this stage of research the interest lay in an actual machine implementation of the theory presented earlier in this chapter. Motivated by the success of this approach, the theory was further developed to solve the general fixed charge problem considered in the next chapter. The program developed for this problem is essentially similar though more sophisticated and is described in detail in that chapter.

Chapter 9

Dichotomizing Procedure for the Linear and Quadratic Fixed Charge Problems.

Introduction

In the fixed charge transportation problem considered in the last chapter, it was observed that the fixed charge variables which can only assume 0-1 values reduce the problem to one of special structure. A similar structure exists for the linear and quadratic fixed charge problems to be defined in the next section, and the concept of the dichotomizing procedure has therefore been extended to obtain similar procedure for their solution. In Section a of this chapter the problem is defined, in Section b an approximate solution of the problem and the upper and lower bounds are established. In Section c a dichotomizing , that is branching, procedure is proposed and in Section d the finiteness of such a procedure is proved. Section e provides two worked examples and the algorithm is formally stated in Section f.

Section g describes the layout and the organization of the program. Section h discusses the results of experiments carried out with example problems and of applying alternative branching heuristics.

a. Statement of the problem

Consider the problem P:

$$\text{Minimize } Z(x,\delta) = \sum_{j=1}^{n} c_j x_j + q_j x_j^2 + d_j \delta_j \qquad (9.1)$$

$$(c_j, q_j, d_j \geq 0)$$

subject to the linear constraints,

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i, \quad i = 1,2,\ldots m \qquad (9.2)$$

$$0 \leq x_j \leq \phi_j \qquad (9.3)$$

and the non-linear constraints,

$$\left.\begin{aligned} \delta_j &= 1 , \text{ if } x_j > 0 \\ \delta_j &= 0 , \text{ if } x_j = 0 \end{aligned}\right\} \qquad (9.4)$$

Let the constraint (9.4) be replaced by

$$0 \leq \delta_j \leq 1, \quad \delta_j \text{ integer}, \qquad (9.5)$$

$$j = 1,2,\ldots n$$

$$x_j \leq \phi_j \delta_j \qquad (9.6)$$

As in Chapter 8 the solution space represented by the constraints (9.2), (9.3), (9.5) is a linear space if the integrality requirement on the δ variables are ignored. The feasible solutions to the problem P are embedded in this space. The problem can be solved approximately by the separable variable technique, in conjunction with an interpolation scheme [3.12]. Following up this concept and applying integer constraints the above can be extended to obtain the global optimum of the problem P. In the knowledge of the author no such scheme has been actually tried.

b. Approximate solution and upper and lower bounds

Let $\bar{P}$ denote the convex problem defined above that is the problem P with the integrality requirement of the constraint (9.5) ignored. Just as Balinski [3.9] shows, in the optimal solution of the problem $\bar{P}$, $x_j \leq \phi_j \delta_j$ are satisfied at the limit that is $x_j = \phi_j \delta_j$ for all j. For suppose the optimal solution to the problem $\bar{P}$ has been obtained, and consider the values taken by variables in this solution. For $\delta_j = 0$ the above is obviously true. For some s such that $\delta_s > 0$ suppose the contrary is true, that is, $x_s < \phi_s \delta_s$. Consider the corresponding cost coefficient $d_s \geq 0$, and a quantity $\delta_s' = \frac{x_s}{\phi_s}$; clearly $\delta_s' < \delta_s$. Therefore in the optimal solution to $\bar{P}$ if δ_s is replaced by δ_s' (δ_s does not interact with any variable other than x_s) then one would obtain a feasible solution to the problem $\bar{P}$ the cost of which is less than the optimal solution by the quantity

$d_s(\delta_s - \delta'_s)$. If $d_s > 0$ this contradicts the assumption of optimality for $d_s = 0$ the corresponding δ_s variable may be excluded from the formulation.

Let an auxiliary problem Q be defined,

$$\text{minimize} \sum_{j=1}^{n} \left(c_j + \frac{d_j}{2}\right) x_j + q_j x_j^2 \qquad (9.7)$$

subject to the constraints (9.2) and (9.3)

Again it can be shown as in [39] that the cost of the optimal solution $(\bar{x}^o, \delta^o)$ to the problem $\bar{P}$, $Z_{\bar{P}}$ is equal to Z_Q the cost of the optimal solution $\bar{x}^o$ to the problem Q. The solution space S_P of the mixed integer problem P is subspace of $S_{\bar{P}}$, the solution space of the problem $\bar{P}$. Further, because of the convexity of the problem $\bar{P}$ it follows that $Z_{\bar{P}} \leq Z_{OPT}$, where Z_{OPT} is the cost of the optimal solution to the problem P. Let $(\bar{x}^o, \delta^o)$ denote the corresponding feasible solution to the original problem P constructed by imposing (9.4) and $Z_p(\bar{x}^o, \delta^o)$ the cost of this solution. Then this must be an upperbound to Z_{OPT}. The following bounds are therefore established

$$Z_Q = Z_{\bar{P}} \leq Z_{OPT} \leq Z_P(\bar{x}^o, \delta^o) \qquad (9.8)$$

C. A branching process and improvement of the lower bound

The concept of 0 - 1 variable branching schemes to solve integer linear programs can be extended to solve the mixed integer quadratic problem P defined in Section 4.

The constraint $0 \leq \delta_s \leq 1$, δ_s integer (9.5) of the problem P enables one to define two subproblems,

a) P ($\delta_s = 1$) which is problem P together with additional constraint $\delta_s = 1$,

b) P ($\delta_s = 0$) which is problem P together with the additional constraint $\delta_s = 0$.

Let $Z_{OPT}\, P(\delta_s = 1)$ denote the cost of the optimal solution to the problem P ($\delta_s = 1$) and $Z_{OPT}\, P(\delta_s = 0)$ the cost of the optimal solution to the problem P($\delta_s = 0$).

Then by direct enumeration it must be true that

$$Z_{OPT} = \min \{Z_{OPT}\ P(\delta_s = 1),\ Z_{OPT}\ P(\delta_s = 0)\} \qquad (9.9)$$

Let $\bar{P}(\delta_s = 1)$ denote the subproblem auxiliary to the subproblem $P(\delta_s = 1)$ with the integer constraint on the rest of the variables $\delta_j (j=1,2,\ldots n \text{ and } j \neq s)$ ignored. The result of Section 5 can be extended to show that in the optimal solution to the problem $\bar{P}(\delta_s = 1)$, $x_j = \phi_j \delta_j$ for all j except j=s. Further, the constraint $0 \leq x_s \leq \phi_s \delta_s$ will always be satisfied, since $\delta_s = 1$ and constraints (9.3) ensure that $x_s \leq \phi_s$.

If one solves the problem $Q(\delta_s = 1)$:

$$\text{Minimize } Z_{Q(\delta_s = 1)}\ (x) = \sum_{j=1}^{n} (\Pi_j x_j + q_j x_j^2) + d_s \delta_s \qquad (9.10)$$

subject to the constraints (9.2) and (9.3) and $\delta_s = 1$, where $\Pi_j = c_j + \frac{d_j}{\phi_j}$ for all j except j=s and $\Pi_s = c_s$, then it can be shown that the cost $Z_{Q(\delta_s = 1)}$ of the optimal solution to this problem must be the same as the cost $Z_{\bar{P}(\delta_s = 1)}$ of the optimal solution to the problem $\bar{P}(\delta_s = 1)$. Let $Z_{Q'(\delta_s = 1)}$ denote the cost of the optimal solution to the problem $Q'(\delta_s = 1)$:

$$\text{Minimize } Z_{Q'(\delta_s = 1)}\ (x) = \sum_{j=1}^{n} (\Pi_j x_j + q_j x_j^2), \qquad (9.11)$$

subject to the constraints (9.2), (9.3). $Q'(\delta_s = 1)$ can be solved by a convex quadratic programming algorithm, and $Z_{Q'(\delta_s = 1)} + d_s = Z_{Q(\delta_s = 1)}$. This establishes a lower bound on $Z_{OPT}\ P(\delta_s = 1)$;

$$Z_{Q'(\delta_s = 1)} + d_s = Z_{Q(\delta_s = 1)} = Z_{\bar{P}(\delta_s = 1)} \leq Z_{OPT}\ P(\delta_s = 1). \qquad (9.12)$$

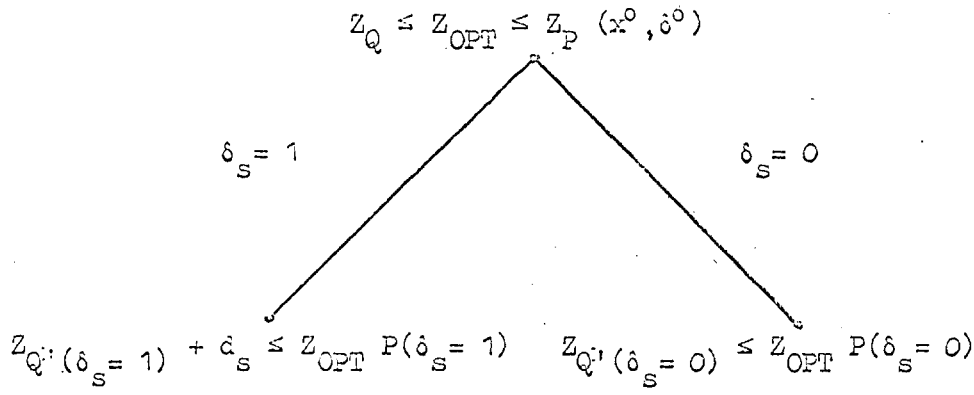
Similarly, let $\bar{P}(\delta_s = 0)$ denote the problem corresponding to the problem $P(\delta_s = 0)$ with the integrality requirement on rest of the variable δ_j ignored. It can again be shown that in the optimum solution of $\bar{P}(\delta_s = 0)$, $x_j = \phi_j \delta_j$ for all j and $x_s = \delta_s = 0$, since $0 \leq x_s \leq \phi_s \delta_s$. In this case it is not necessary to consider an itermediate $Q(\delta_s = 0)$ problem explicitly. If one solves the problem $Q'(\delta_s = 0)$:

$$\text{Minimize } Z_{Q'(\delta_s = 0)}\ (x) = \sum_{j=1}^{n} (\Pi_j x_j + q_j x_j^2), \qquad (9.13)$$

where $\Pi_j = c_j + \frac{d_j}{\phi_j}$ for all j, subject to the constraints (9.2) and (9.3) and the additional constraint $x_s = 0$, then it can be shown trivially that $Z_{Q'}(\delta_s = 0)$, the cost of the optimal solution to this problem, will be the same as $Z_Q(\delta_s = 0) = Z_P(\delta_s = 0)$, the cost of the optimal solution to the problem $\bar{P}(\delta_s = 0)$. The following bound is therefore established

$$Z_{Q'}(\delta_s = 0) = Z_{\bar{P}}(\delta_s = 0) \leq Z_{OPT}\, P(\delta_s = 0) \tag{9.14}$$

It is of interest to note that (9.12) and (9.14) can be established intuitively by arguments based on principle of optimality of dynamic programming. The relations (9.8), (9.9), (9.12), (9.14) are illustrated in the two branches of Fig. (9.1)



$$Z_{OPT} = \min \{Z_{OPT}\, P(\delta_s = 1),\ Z_{OPT}\, P(\delta_s = 0)\}$$

Fig. 9.1

So far, starting from the problem P, only a single step in the branching process has been discussed, but these arguments can be extended, in general, to any depth of branching. Further, the branching can be started on any of the variables δ_j of the problem P. However, it is of advantage to start from the optimal solution x^o of the problem Q, choose a variable $x^o{}_s > 0$ and branch on its corresponding fixed charge variable δ_s. It will be shown in the next section that $Z_{Q'}(\delta_s = 1) + d_s \geq Z_Q$ and $Z_{Q'}(\delta_s = 0) \geq Z_Q$, that is, whenever branching is undertaken the lower bounds are improved or at worst held at the same value. The lower bounds shown at each node of the bifurcating tree which is being

constructed are local to the branches emanating from it. However, the currently available minimum cost feasible solution constitutes the global upper bound at any stage of the development of the tree. The search therefore consists in defining new subproblems of types(Q.11) and(Q.13), and solving for possibly better lower bounds(Q.12) and(Q.14); the corresponding feasible solutions to the problem P are checked for better upper bounds. The search is terminated when all the local lower bounds at the developed nodes become equal to or greater than the current best upper bound.

d. Behaviour of the lower bound and establishment of finiteness

Let the original problems P and Q be relabelled P^o and Q^o. The variables shown in Fig9.2, illustrating further details of the branching process, are explained below.

Let

δ_{si} denote the fixed charge variable which is being dichotomized at the current level i,

f_c^i denote the cumulative fixed charge at the current node corresponding to those δ_s which are set equal to 1 (the relation by which f_c^i is updated is given later(Q.20)),

$P^i(\delta_{si}=1)$, $P^i(\delta_{si}=0)$ represent the two subproblems of type P emerging from the predecessor problem P^{i-1},

$Q^i(\delta_{si}=1)$, $Q^i(\delta_{si}=0)$ represent the two derived subproblems of type Q which may be considered as emerging from the predecessor problem Q^{i-1}.

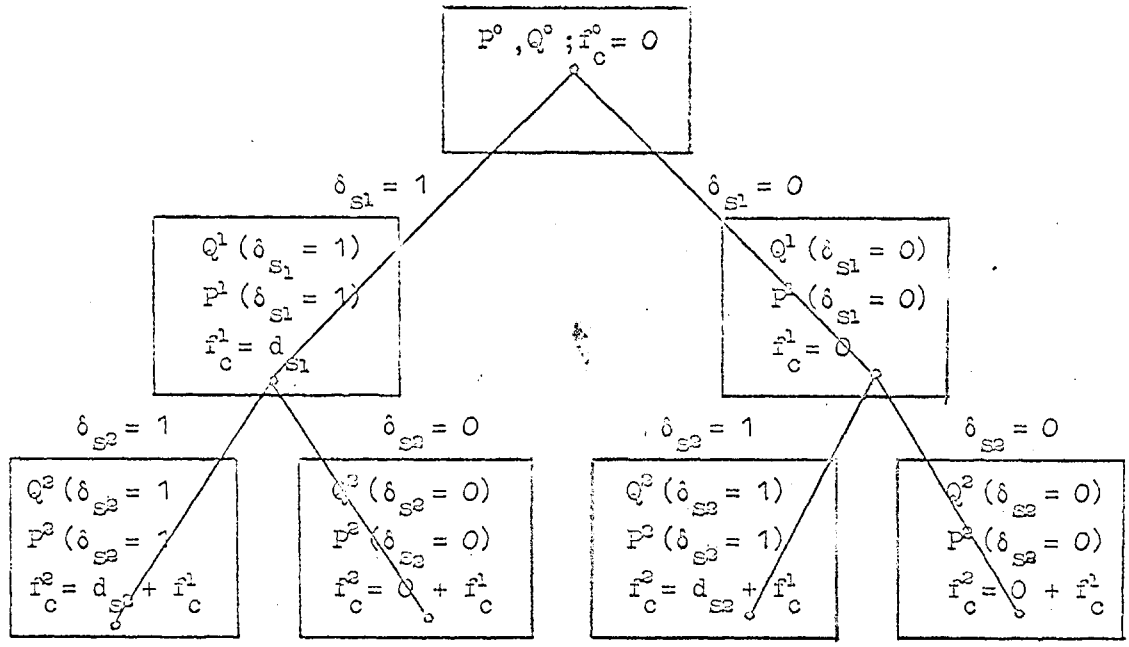
The actual problems solved are the Q' problems(Q.11) and(Q.13) which are conceptually the same as the corresponding Q problems. In the above notation the index i denotes the level of any node of the tree, as illustrated in Fig. 9.2. To avoid cumbersome notation s will not be subscripted in the subsequent

derivation, and moreover the distinction between the $\delta_s = 1$ and $\delta_s = 0$ type problems will be dropped. Let Z_{Qi} denote the cost of the optimal solution x^i to the problem Q^i then x^{i+1} denotes the optimal solution of the problem $Q'^{i+1}(\delta_s = 1)$, (Q.11), and $Z_{Q'i+1}(\delta_s = 1)$ its cost. The costs of these two adjacent optimal solutions are given by,

$$Z_{Qi} = x_s^i\left(\frac{a_s}{\phi_s} + c_s\right) + (x_s^i)^2 a_s + C^i \qquad (Q.15)$$

and

$$Z_{Q'i+1}(\delta_s = 1) = x_s^{i+1} c_s + (x_s^{i+1})^2 a_s + C^{i+1} \qquad (Q.16)$$



$$Z_{Qi}(\delta_{si} = 0) = Z_{Q'i}(\delta_{si} = 0) + f_c^i$$

$$Z_{Qi}(\delta_{si} = 1) = Z_{Q'i}(\delta_{si} = 1) + f_c^i$$

and $Z_{Qi}(\delta_{si} = 0) \leq Z_{OPT}\ P^i(\delta_{si} = 0)$ and $Z_{Qi}(\delta_{si} = 1) \leq Z_{OPT}\ P^i(\delta_{si} = 1)$, i=0,1,...nxm.

In general the above relations hold at any node of the tree.

Fig. Q.2.

The general expression for the lower bounds on both the branches can thus be written $Z_{Q\,i+1} = Z_{Q\,i+1} + r_c^{i+1}$, and these have already been shown to be at least as good as Z_{Qi}.

Thus there is no decrease and potential increase in the lower bounds down the branches of the tree. To prove that the algorithm terminates in finite number of steps one needs only to show that the growth of the tree is finite. At worst, it may be necessary to branch on all the n variables δ_j, j= 1,2,...n. In this case the total number of nodes generated will be $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$, a finite number. Although this is a frighteningly large number, in the actual solution of the problems, because of the facility of pruning the tree whenever the global upper bound is exceeded by the local lower bound at a node, the tree developed is of much reduced size.

e. Examples

So far no particular method for the solution of the convex quadratic programming Q' problems has been suggested. In fact any of the various quadratic programming algorithms may well be applied. However, as is illustrated in the following examples, one desirable property of the algorithm is that it should be able to handle parametric variation of the linear cost terms or the right hand side of the constraints, that is, the ability to resolve dual and primal infeasibilities once these are introduced by the parameter changes. A modification of the simplex method of quadratic programming of Van-de-Panne and Whinston [2.13] in which the symmetry/skew symmetry property of the quadratic programming tableau is never destroyed, has been developed. Some features of this algorithm are given in ~~the Appendix~~ Chapter 5.

Consider first the problem P^0:

Minimize $x_1 + 3x_1^2 + x_2 + 5x_2^2 + 10\delta_1 + 15\delta_2$ (9.21)

subject to,

$$\left.\begin{aligned} 2x_1 + 5x_2 &\geq 10 \\ 2x_1 + x_2 &\geq 4 \\ 0 \leq x_1 &\leq 10 \\ 0 \leq x_2 &\leq 10 \end{aligned}\right\} \qquad (3.22)$$

$$\left.\begin{aligned} &0 \leq \delta_1 \leq 1,\ 0 \leq \delta_2 \leq 1,\ \delta_1,\ \delta_2 \text{ integer} \\ &\text{and } 0 \leq x_1 \leq 10\delta_1,\ 0 \leq x_2 \leq 10\delta_2 \end{aligned}\right\} \qquad (3.23)$$

As in Section 5 the approximating problem Q^0 can be written:

$$\text{Minimize } 2x_1 + 3x_1^2 + 2{\cdot}5x_2 + 5x_2^2,$$

subject to (3.22). This problem is set up in the symmetric/skew symmetric quadratic programming tableau, tableau 1. Applying the quadratic programming algorithm, the optimal solution shown in tableau 2 is obtained. Constructing from this the solution (x^0, δ^0), (3.4), leads to the lower and upper bounds to the optimal solution,

$$Z_{Q^0} = 22{\cdot}137 \leq Z_{OPT} \leq Z_{P^0} = 43{\cdot}687,$$

at the root node of the tree.

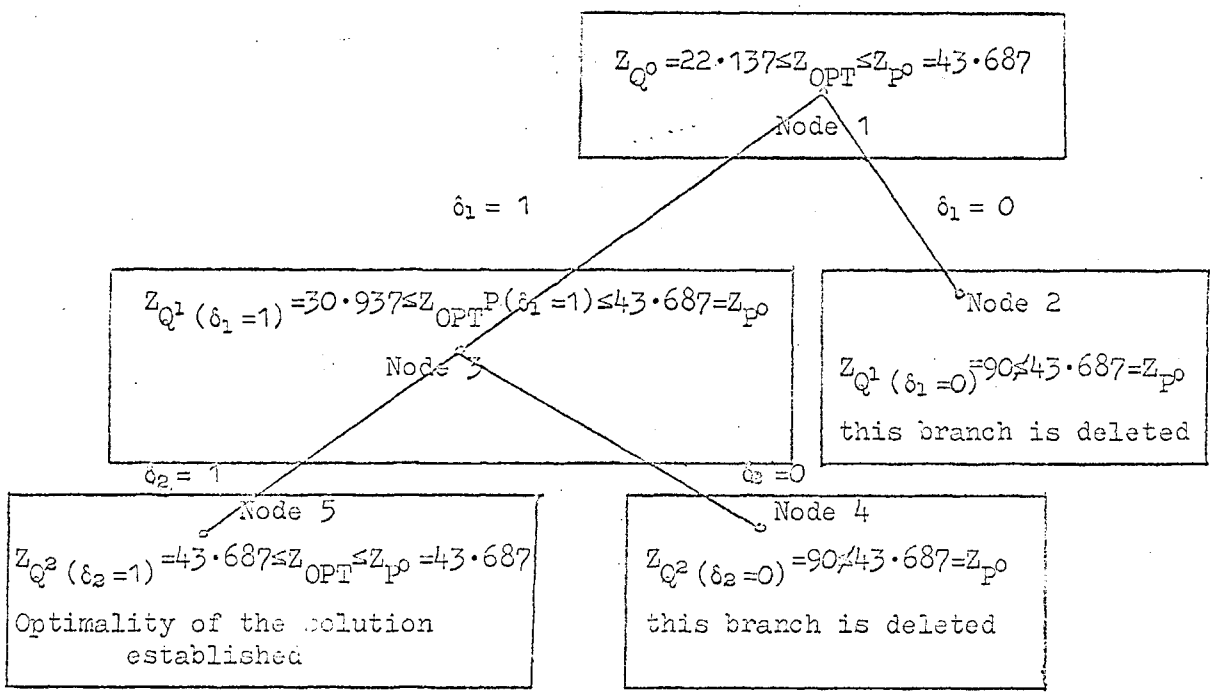


Fig. 9.3a

		x_0	$-x_1$	$-x_2$	$-u_3$	$-u_4$	$-u_5$	$-u_6$
$2f(x) =$	u_0	0						
	u_1	2	-6					
	u_2	2·5	0	-10				
	x_3	-10	-2	-5	0			
	x_4	-4	-2	-1	0	0		
	x_5	10	1	0	0	0	0	
	x_6	10	0	1	0	0	0	0

Tableau 1

		x_0	$-u_1$	$-u_2$	$-x_3$	$-x_4$	$-u_5$	u_6
$2f(x) =$	u_0	-44·375						
	x_1	1·250	0					
	x_2	1·500	0	0				
	u_3	3·188	-0·125	0·250	-0·719			
	u_4	1·563	0·625	-0·250	1·094	-2·969		
	x_5	8·750	0	0	-0·125	0·625	0	
	x_6	8·500	0	0	0·250	-0·250	0	0

Tableau 2

From Node 1 two subproblems $P^1(\delta_1 = 1)$ and $P^1(\delta_1 = 0)$ are defined. Let $Q'^1(\delta_1 = 1)$ and $Q'^1(\delta_1 = 0)$ be the corresponding Q problems actually investigated. It is not necessary to start with new set-up tableaus for each of these problems. The problem $Q'^1(\delta_1 = 0)$ is the same as the problem Q^0 with the additional constraint $x_1 = 0$, since $\delta_1 = 0$, and may be represented simply by altering the upper bound

$x_1 \leq 10$ to $x_1 \leq 0$. Hence, in the optimal tableau 2 the value $x_6 = \alpha_{50} = 8{\cdot}75$ becomes $x_6 = \alpha_{50} = -1{\cdot}25$, giving a primal infeasibility. The dual algorithm now leads to the optimal solution of Q'^1 $(\delta_1 = 0)$ with cost, $Z_{Q'^1(\delta_1=0)} = Z_{Q^1(\delta_1=0)} = 90$. As this is larger than the current best solution, the branch is deleted. The problem $Q'^1(\delta_1 = 1)$ of the other branch is next investigated; this problem is again the same as problem Q^0 apart from the linear cost term $\Pi_1 = 2$, which is altered to $\Pi_1 = 1$ and f_c^1 is set equal to 10. The problem is solved parametrically and the lower bound $Z_{Q^1(\delta_1=1)} = Z_{Q'^1(\delta_1=1)} + f_c^1 = 30{\cdot}937$ obtained (Node 3, Fig.9.3). Note that branching to the right $\delta_s = 0$ always produces primal infeasibility, whereas branching to the left $\delta_s = 1$ may or may not produce dual infeasibility. The branching process is continued from the Node 3 and after developing Node 4 and Node 5 the optimality of the solution $x_1 = 1{\cdot}25$, $x_3 = 1{\cdot}50$ with $Z_{OPT} = 43{\cdot}687$ is established.

As a problem for which the first approximate solution is not the optimal solution a linear fixed charge problem of production scheduling is now considered. A manufacturer has to meet the demands 90, 125, 140, in three successive months. The set up cost is £300 per production run and the inventory carrying charge is £2 / month / item . The problem is taken from Hadley [1·8, p 141] and is formulated in a slightly different form. Let x_1, x_2, x_3 denote the quantitites produced in the three time periods 1,2,3 and let δ_1, δ_2, δ_3 denote the corresponding fixed charge variables. From the data one can compile Table 9.1.

Period	Backward Totals	Demands	Forward Totals
1	355	90	90
2	265	125	215
3	140	140	355

Table 9.1

Assuming there are no inventories at start or at finish, the problem can be clearly formulated as,

$$P^0: \text{ Minimize } Z = 2(x_1 - 90) + 2(x_1 + x_2 - 215) + 300(\delta_1 + \delta_2 + \delta_3)$$

$$= -610 + 4x_1 + 2x_2 + 300\,(\delta_1 + \delta_2 + \delta_3) \qquad (9.24)$$

subject to

$$\left.\begin{aligned} x_1 &\geq 90 \\ x_1 + x_2 &\geq 215 \\ x_1 + x_2 + x_3 &= 355 \end{aligned}\right\} \qquad (9.25$$

$$\left.\begin{aligned} x_1 &\leq 355\,\delta_1 \\ x_2 &\leq 265\,\delta_2 \\ x_3 &\leq 140\,\delta_3 \end{aligned}\right\} \qquad (9.26)$$

Hence, the auxiliary problem Q^0 is

$$\text{Minimize } Z_{Q^0} = x_1\left(4 + \frac{300}{355}\right) + x_2\left(2 + \frac{300}{265}\right) + x_3\,\frac{300}{140} \qquad (9.27)$$

subject to (25). The solution tree is illustrated in Fig. 9.3a.

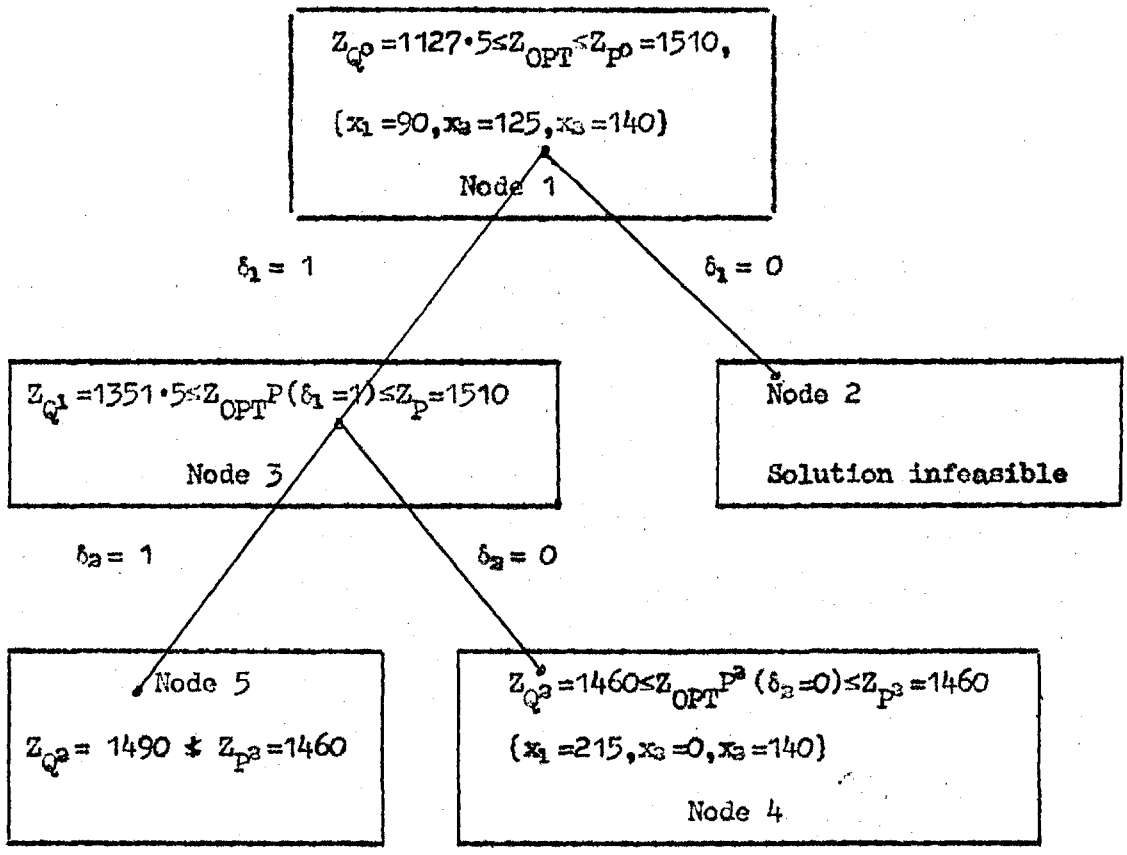


Fig. 9.3a

Initially the global upper bound is $Z_{p0} = 1510$ (Node 1). After developing Node 4, and checking the corresponding feasible solution, a better global upper bound $Z_{p0} = 1460$, $(x_1 = 215,\ x_2 = 0,\ x_3 = 140)$ is obtained. The optimality of this solution is established after developing Node 5.

f. Statement of the algorithm

Let,

Store — denote the backing store that will hold any number of subproblems corresponding to the nodes of the tree,

S_i — denote an intermediate store that will hold one of the two successor problems generated when a problem is dichotomized,

chprob — denote a procedure for choosing the next problem to be considered from the tree of subproblems,

chvar — denote a procedure to choose the variable x_s on which branching is undertaken, after a problem has been chosen. The index s must be such that

$$s \in A_1 \text{ and not } A_2 \qquad (9.28)$$

where A_1 represents the set of indices of the basic x variables in the current optimal tableau and A_2 the set of indices of the variables already dichotomized. In general, P problems refer to the integer problems which need to be solved and Q'problems to the approximating problems which are actually investigated to determine lower bounds to P problems. The steps of the algorithm are outlined below.

Step 1 — Given the problem P^0 derive the problem Q^0.

Step 2 — Solve the problem Q^0 and obtain Z_{Q^0} and Z_{P^0}.

Step 3 — By the procedure chvar obtain the index 's' of the variable on which to dichotomize and produce two subproblems (successor problems) $Q^{i+1}(\delta_s=1)$, and $Q^{i+1}(\delta_s= 0)$.

Step 4 — Store $Q^{i+1}(\delta_s= 1)$ in S_i and goto step 6.

Step 5 — Bring back $Q^{i+1}(\delta_s= 1)$ to the main tableau area; S_i is now empty.

Step 6 — From the optimal tableau of Q^i solve Q^{i+1} parametrically.

Logical Step 1 — If the cost of the corresponding solution to the P problem is cheaper than the best solution obtained so far then overwrite the current best solution.

Logical Step 2 — If the new local lower bound is not less than the cost of the current best solution, then goto logical step 3 (i.e., this problem will be overwritten and hence the branch deleted).

Step 7 — Store the problem matrix and other information in 'Store'; this should incorporate a suitable stack mechanism with a pointer.

Logical Step 3 — If S_i is full then goto step 5.

Logical Step 4 — If 'Store' is empty then goto Exit.

Step 8 — Fetch a problem by the procedure chprob, from 'Store' and goto step 3.

Exit: — Print the current best solution which is the optimal solution.

8. The Organization of the Program and the heuristics of branching

The listing of the program written in Algol is provided in [illegible] Appendix 2, where the input format is also defined. The layout of the store (Fig. 9.4) and the important procedures are explained below. (Unless otherwise specified the arrays and the variables should be considered as of type integer).

ub — denotes the upper bound of the immediate access store used.

Store [1:ub] <real array> — denotes the main area of the store (immediate access/core) where subproblems are stored.

trlength — denotes the number of blocks of core store used and also the length of the record to be transferred to magnetic tape when necessary. Thus ub=trlength × 512.

length — denotes the size of a tableau plus a few other information in terms of an one dimensional array, which defines a unit of problem storage.

lim — denotes the upper limit of the number of problems which can be stored in 'Store'. lim=entier (ub/(length + 1)).

avlist [0:lim] — denotes an array of pointers to the problems in 'Store'.

avlist [0] — points to the currently filled up problem location, i.e. the last entry in the push down list. The array avlist [0:lim] is itself written up at (or read in from) the end of Store [1:ub] when transfer to or from magnetic tape takes place. This is done to preserve the ordering of the problems when transferred to or from magnetic tape.

avlist [lim] defines the intermediate store S_i and points to the problem location shown by the shaded area.

The pointers avlist [i] are set up by the following statement:

for i:= 1 step 1 until lim do avlist [i]:= (i-1) × length + 1;

mkr <boolean> indicates if S_i is full or empty.

empty <boolean> indicates if the last location avlist [lim-1] in the push down list is full or empty.

send is a procedure to send a problem to the 'Store'.

fetch is a procedure to fetch a problem from the 'Store'.

logic is a procedure which organizes transfer to magnetic tape if and when necessary.

invlogic is a procedure which organizes transfer from magnetic tape if and when necessary.

chprob is a procedure to choose the next problem from the tree of subproblems. It is controlled by a global variables 'choose prob'.

chvar is a procedure to determine the index of the variable to dichotomize. It is controlled by a global variable 'choose var'.

Choose prob can assume values 1, 2, 3; choose var can assume values 1, 2.

choose prob = 1, the next problem is chosen by the LIFO (last in first out) scheme.

choose prob = 2, the next problem is chosen randomly from the immediate access 'Store'.

choose prob = 3, the next problem has the lowest lower bound of all the problems in the immediate access 'Store'.

choose var = 1, the next variable is chosen at random from the set A_1 and not A_2 defined in Section f (9.28).

choose var = 2, the next variable is chosen by the largest error reduction scheme is $e_s = \max_{i \in A_1 \text{ and not } A_2} d_i \frac{(\phi_i - x_i)}{\phi_i}$

The experiments carried out with the six heuristic rules so defined (cf Tables 9.4, 9.5, 9.6) are discussed in the next section.

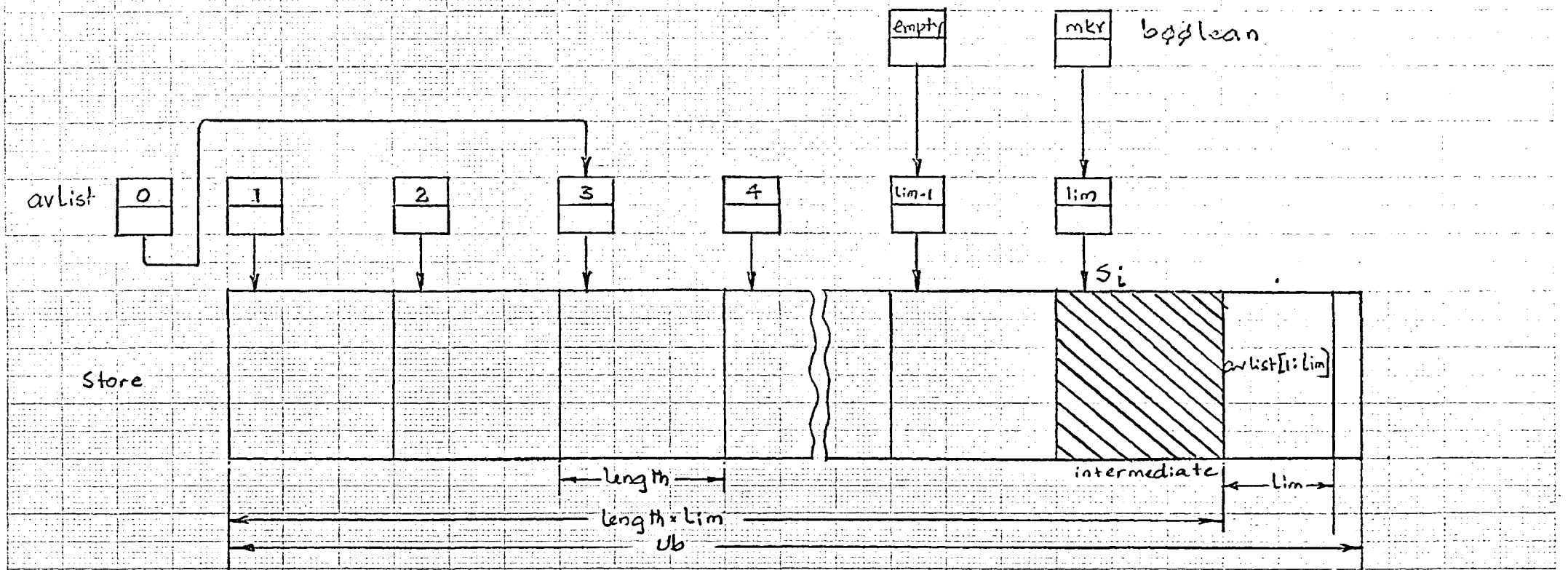

empty
mkr
boolean
avlist
0
1
2
3
4
lim-1
lim
Si
Store
avlist[1:lim]
length
intermediate
lim
length×lim
Ub

Fig. 9.4

4. Results of the experiments with sample set of problems and conclusions.

In all 27 linear fixed charge problems and 17 quadratic fixed charge problems have been solved by the algorithm described in Section 5 and applying each of the six heuristic rules in turn described in Section 3.

Ten linear Production Scheduling problems (P.S.P) have been obtained from the data supplied in [3·12], however, some of the cost coefficients have been slightly altered. These are actually two, 9 period problems, from which subproblems have been extracted by considering 8, 7, 6, 5, periods respectively. The ten problems are specified by the data given in tables 3·2 and 3·3. The corresponding integer linear programming problems are formulated as in the example discussed in Section 2. In the Appendix these problems numbered 18 - 27 are presented in the format specified by the code.

Period No.	Demands	R.H.S. Forward Totals	Backward Totals (Bounds) for the quantities produced in these periods				
			9	8	7	6	5
1	90	90	900	840	690	560	500
2	125	215	810	750	600	470	410
3	140	355	685	625	475	345	285
4	100	455	545	485	335	205	145
5	45	500	445	385	235	105	45
6	60	560	400	340	190	60	
7	130	690	340	280	130		
8	150	840	210	150			
9	60	900	60				
	Fixed charge d_j = 300 units per production run inventory carrying charge 2 units / item / period.						

Table 3·2.

Period No.	Demands	R.H.S Forward Totals	Backward Totals (Bounds) for the quantities produced in these periods				
			9	8	7	6	5
1	80	80	830	730	605	505	455
2	100	180	750	650	525	425	375
3	125	305	650	550	425	325	275
4	100	405	525	425	300	200	150
5	50	455	425	325	200	100	50
6	50	505	375	275	150	50	
7	100	605	325	225	100		
8	125	730	225	125			
9	100	830	100				
Fixed charge d_j = 250 units per production run inventory carrying charge = 2 units / item / period.							

Table 9·3.

The column for Forward Totals provides the requirement vector of the inequality constraints and the column for the Backward Totals provides the upper bounds on the structural variables i.e., the quantities produced in these periods. The objective function and the constraints, (9·1) to(9·3), for the 17 quadratic fixed charge problems have been generated by the following rules

$$0 \leq a_{ij} \leq 20$$

$$1 \leq c_j \leq 20$$

$$1 \leq d_j \leq 200$$

$$200 \leq b_j \leq 400$$

$$d_j \leq \phi_j \leq 500$$

$$q_j = 2\,\frac{(100 - w_j)}{100} \cdot \frac{c_j}{\phi_j} \quad \text{where } 1 \leq w_j \leq 10$$

These coefficients for the appropriate ranges of i and j, have been extracted such that these are integers drawn from an uniform distribution between the upper and lower bounds specified by the above equations [3·13]. The sparsity of the constraint matrix a_{ij} has been controlled so that these problems are 33%, 50%, 66% sparse. To obtain 33% sparse matrix, for instance, uniformly distributed random digits in the range 1 to 30 were looked up [3·14] and the integers in the range 20 to 30 replaced by zero. The 17 random linear problems (R.L.P.) problems were obtained by using the same constraint matrices as those of the random quadratic problems (R.Q.P.), and further setting $q_j = 0$ for all j. The q_js of the (R.Q.P.) problems have often been suitably scaled or altered so that the solutions of the linear and quadratic problems become different. These problems are numbered 1 to 17 and presented in the Appendix B.

The results of the experiments carried out with these examples are shown in the tables 9·4, 9·5, and 9·6. From the tables 9·4, 9·5, and 9·6, for fixed values of the number of variables (and this restricts the sample size to 7 or less in each category) the means of the ln (log natural) of the number of computing instructions and the means of the ln of the maximum store used, were extracted and the tables 9·7, 9·8, and 9·9 were compiled. The data in the tables 9·7, 9·8, 9·9 were plotted as curves of the mean ln (number of computing instructions), and mean ln (maximum store used) against the number of variables, Fig. 9·5, 9·6, 9·7, and these plots show a roughly linear relationship between the independent and the dependent variables. Other functions like square root etc. were tried before deciding on the ln function. Since, computing cost in a multiprogramming machine is proportional to the product of the number of computing instructions and storage used, the ln function is a natural function to use. From the Figs, 9·5, 9·6, 9·7 one observes that the choice of the variables is a more important criterion in determining which heuristic rule to use for branching [note the legend

Fig. 9·5.a. gives symbols with central dots and hatches to represent the two choices]. Thus the largest error reduction scheme (central dots) is almost always a superior strategy to adopt.

To investigate the best rules for choosing a problem from the store, and to consider the effect of sparsity, further analysis of the data was done as follows. The observations for all the heuristic rules were processed by a two variable regression program. The results in general showed high degree of correlation between ln (number of computing instructions) vs number of variables, ln (maximum store used) vs no of variables, and no significant correlation between sparsity and ln (number of computing instructions) and ln (maximum store used). However, high partial correlation between ln (number of computing instructions) and ln (maximum store used) for fixed number of variables was observed. The results for the two most promising heuristics, Rule-2 and Rule-6 are illustrated in the Tables 9·10, 9·11, 9·12.

The high partial correlation between ln (number of computing instructions) and ln (maximum store used), given the number of variables implies that these two quantities move together after correction for dependence on the number of variables. Thus either would provide a measure of computing effort. Because it is a measure of cost, in fact the sum of these two quantities is taken as a variable against which the performance of the two heuristic rules 2 and 6 can be compared. The regression parameters calculated for the two heuristic rules are shown in Tables 9·13 and the regression curves shown in Figs, 9·8, 9·9 and 9·10, with the observations plotted as well. The fact that the plots for the two rules are mostly paired, reflects that the between-problem variation is more important than the variance between these two heuristics. To eliminate the between problem variance a new quantity Z_i is introduced

such that

$$Z_i = x_i(6) - x_i(2)$$

where, $x_i(6)$ = ln (number of computing instructions x maximum store used) for the heuristic rule no. 6.

$x_i(2)$ = ln (number of computing instructions x maximum store used) for the heuristic rule no. 2.

These results are plotted in Fig. 9·11 and the means and the standard deviations for the three classes of problems considered are shown in Table 9·14. The test of the heuristic rule no. 2 performing better than heuristic rule no. 6 is given by the following t-statistic

$$t_{n-1} \simeq \frac{\bar{Z}}{\sqrt{\frac{1}{n} S_Z^2}}$$

where $\bar{Z}$ = sample mean of z_i; S_Z = sample standard deviation of z_i. r = no of observations leading to n-1 degrees of freedom. Table 9·14 summarizes the significance levels for these problems. It is perhaps worth noting at this point that there is a definite pattern in the distribution of the optimum solution between the initial upper and lower bounds, and is a property of the problem as is illustrated by Fig. 9·12. This property may be employed to obtain a suitable strategy for a cut-off rule, the object being to be able to prune the unprofitable branches of the tree early in the search.

Lastly to justify the use of the technique developed in this chapter in preference to the integer programming approach Chapter 7 consider the problem 5 (R.Q.P.) given in Appendix 13. If formulated as an integer quadratic program (Appendix 14) it can be solved by the technique described in Chapter 7. Allowing for the same initial storage (although the method of Chapter 7 will require much larger storage, because of the higher dimension of the problem) the figures for the computing instructions and the number of iterations are compared in table 9·15.

Legend for Tables 9.4, 9.5, 9.6:

a_1 = No. of nodes or subproblems investigated.

a_2 = Maximum number of problems stored in the STACK (store) at any one time - a measure of the maximum store used.

a_3 = Total number of pivot operations (iterations) required to solve the problem. [includes both double and single pivot].

a_4 = Ratio of the number of computing instructions required in the tree search phase to the number of computing instructions required to arrive at the first solution.

a_5 = log (natural) of the total number of computing instructions required to solve the problem.

pr = Number of computing instructions required to obtain the first approximate solution.

qr = Indicates the location of the optimal solution between the initial upper and lower bounds. $qr = \frac{Zopt - Z_Q}{Z_P - Z_Q}$, where Z_P, Zopt, Z_Q are defined below.

sr = The sparsity of the constraints matrix given by the ratio: (Total number of zero elements)/(Total number of elements.)

Z_P = The cost of the first approximate feasible solution to the problem - initial upper bound.

Zopt = The cost of the optimal solution to the problem.

Z_Q = The cost of the optimal solution to the first approximating problem - the lower bound.

V = The number of variables of the problem, for the class of problems considered. This equals the number of constraints.

Choice of Problem and choice of variable together define six heuristic rules as explained in Section 3.

GM:SA

Table 9.4

Choice of Problem	LAST IN FIRST OUR SCHEME; Choose Prob=1		RANDOM CHOICE SCHEME; Choose Prob=2		Best (lowest) Lowerbound scheme; Choose prob=3		Problem Constants
Choice of Variable	Random Choice choose var=1	Largest error choice choose var=2	Random Choice choose var=1	Largest error choice choose var=2	Random Choice choose var=1	Largest error choice Choose var=2	
Heuristic Rule No.	1	2	3	4	5	6	
Problem No.	a_1, a_2, a_3, a_4, a_5	a_1, a_2, a_3, a_4, a_5	a_1, a_2, a_3, a_4, a_5	a_1, a_2, a_3, a_4, a_5	a_1, a_2, a_3, a_4, a_5	a_1, a_2, a_3, a_4, a_5	pr, qr, sr, Zp, Zopt, ZQ, V,
1	20, 2, 8, 4·707, 5·8693	8, 1, 5, 2·347, 5·1818	14, 2, 5, 3·250, 5·5134	8, 1, 5, 2·273, 5·1762	10, 1, 5, 2·592, 5·2933	8, 1, 5, 2·303, 5·1762	76, 0·90, 0·5, 7040, 6920, 5825, 5
2	22, 3, 13, 5·243, 6·3321	14, 2, 9, 3·557, 5·9375	46, 4, 24, 10·299, 7·0067	14, 2, 9, 3·533, 5·9375	30, 6, 19, 7·215, 6·6516	14, 2, 9, 3·523, 5·9375	107, 0·81, 0·5, 9700, 9400, 8092, 6
3	76, 4, 33, 12·810, 7·6746	46, 3, 28, 8·720, 7·2910	74, 5, 29, 12·210, 7·6217	46, 4, 28, 8·726, 7·2910	72, 12, 31, 12·137, 7·6210,	44, 9, 26, 8·341, 7·2407	167, 0·76, 0·5, 12900, 12240, 10163, 7
4	96, 4, 56, 16·299, 8·2033	96, 4, 63,17·193, 8·2522	98, 7, 53, 16·250, 8·2003	94, 5, 61, 16·710, 8·2282	218, 17, 80, 33·867, 8·9388,	92, 15, 59, 16·723, 8·2290	224, 0·80, 0·5, 16260, 15600, 12937, 8
5	190, 7, 80, 29·615, 8·9683	102, 4, 50,16·692, 8·3986	148, 13, 76, 24·528, 8·7797	96, 6, 47, 15·744, 8·3402	138, 13, 60, 22·598, 8·7071,	96, 13, 47, 16·226, 8·3703	266, 0·81, 0·5, 19860, 19200, 16425·4, 9
6	16, 2, 5, 3·635, 5·6021	8, 1, 3, 2·041, 5·0304	20, 2, 7, 4·608, 5·8377	8, 1, 3, 2·027, 5·0239	20, 3, 7, 4·474, 5·8319,	8, 1, 3, 2·013, 5·0304	75, 0·91, 0·5, 6080, 5980, 5028·5, 5
7	20, 2, 8, 4·274, 6·1181	14, 2, 9, 3·491, 5·9375	20, 3, 14, 5·075, 6·3008	14, 2, 9, 3·523, 5·9375	46, 8, 19, 9·617, 6·9373,	14, 2, 9, 3·566, 5·5402	106, 0·85, 0·5, 8400, 8200, 7038·9, 6
8	68, 5, 34, 12·000, 7·6039	48, 4, 30, 9·240, 7·3428	106, 9, 43, 17·557, 7·9841	46, 5, 28, 8·784, 7·2917	64, 10, 26, 10·725, 7·4911,	44, 9, 26, 8·335, 7·2399	167, 0·78, 0·5, 11220, 10720, 8986·7, 7
9	84, 4, 53, 14·661, 8·0974	92, 4, 61,16·464, 8·2131	110, 6, 52, 17·482, 8·2731	88, 8, 57, 15·643, 8·1622	84, 15, 51, 14·746, 8·1029,	86, 15, 55, 15·473, 8·1513	224, 0·82, 0·5, 14140, 13640, 11393, 8
10	232, 6, 105, 33·548, 9·1899	144, 5, 89,23·185, 8·8204	216, 13, 79, 30·079, 9·0774	136, 7, 81, 21·637, 8·7513	206, 15, 84, 30·209, 9·0851,	132, 13, 79, 21·692, 8·7540	292, 0·81, 0·5, 17560, 16960, 14423·4, 9

Ten Linear Production Scheduling problems described in Pg.32

(P.S. problems)

* For legend see Pg 192

= Due to the formulation Pg. Zp, Zopt, ZQ are different from the true values by a constant.

† The problems and the solutions are given in appendix 13.

Table 9.5

Choice of Problem	LAST IN FIRST OUT SCHEME; Choose Prob=1		RANDOM CHOICE SCHEME; Choose Prob=2		Best (lowest) Lowerbound scheme; Choose prob=3		Problem Constants
Choice of Variable	Random Choice choose var=1	Largest error choice choose var=2	Random Choice choose var=1	Largest error choice choose var=2	Random Choice choose var=1	Largest error choice choose var=2	
Heuristic Rule No.	1	2	3	4	5	6	
Problem No.	a_1, a_2, a_3, a_4, a_5,	a_1, a_2, a_3, a_4, a_5,	a_1, a_2, a_3, a_4, a_5	a_1, a_2, a_3, a_4, a_5	a_1, a_2, a_3, a_4, a_5	a_1, a_2, a_3, a_4, a_5	pr, qr, sr, Zp, Zopt, ZQ, V,
1	10, 2, 7, 2·895, 5·3982	8, 1, 4, 2·038, 5·0876	12, 1, 10, 3·487, 5·6348	8, 1, 4, 2·037, 5·0999	14, 2, 10, 3·873, 5·7269	8, 1, 4, 2·076, 5·1060	76, 0·98, 0·33, 1683·3, 1674·75, 1234·49, 5
2	12, 1, 8, 3·395, 5·5568	8, 1, 5, 2·215, 5·1762	10, 1, 6, 2·658, 5·3519	8, 1, 5, 2·200, 5·1818	8, 1, 5, 2·244, 5·1762	8, 1, 5, 2·241, 5·1818	76, 0·65, 0·33, 1578·6, 1305·2, 802·4, 5
3	8, 1, 2, 1·651, 4·9698	10, 1, 1, 1·733, 5·0626	8, 1, 2, 1·556, 4·9558	10, 1, 1, 1·753, 5·0563	8, 1, 2, 1·538, 4·9488	10, 1, 1, 1·703, 5·0563	86, 0·69, 0·50, 2681·8, 2517·1, 2139·9, 5
4	4, 1, 3, 2·000, 4·6444	4, 1, 3, 1·998, 4·6441	4, 1, 3, 1·998, 4·6640	4, 1, 3, 1·989, 4·6641	4, 1, 3, 2·000, 4·6444	4, 1, 3, 2·000, 4,6444	51, 1, 0·50, 1259·7, 1259·7, 906·6, 5
5	6, 1, 8, 2·359, 5·2258	10, 1, 11, 3·392, 5·5984	6, 1, 8, 2·329, 5·2204	10, 1, 11, 3·380, 5·5910	6, 1, 8, 2·346, 5·2204	10, 1, 11, 3·436, 5·5947	78, 1, 0·50, 1465·9, 1465·9, 926·6, 5
6	4, 1, 3, 1·661, 4·6540	4, 1, 3, 1·545, 4·6444	4, 1, 3, 1·545, 4·6444	4, 1, 3, 1·576, 4·6540	4, 1, 3, 1·522, 4·6444	4, 1, 3, 1·537, 4·6540	62, 1, 0·60, 691·3, 691·3, 552·5, 5
7	6, 1, 6, 2·587, 5·1060	6, 1, 6, 2·439, 5·0938	6, 1, 6, 2·424, 5·0814	6, 1, 6, 2·455, 5·0938	6, 1, 6, 2·368, 5·0876	6, 1, 6, 2·455, 5·0999	63, 1, 0·60, 4194·5, 4194·5, 3577·8, 5
8	14, 2, 15, 5·086, 6·5709	8, 1, 8, 2·735, 5·9989	8, 1, 8, 2·740, 5·9963	8, 1, 8, 2·779, 6·0039	16, 3, 23, 6·418, 6·8448	8, 1, 8, 2·760, 6·0014	140, 0·38, 0·33, 1484·5, 1096·2, 855·6, 8
9	8, 1, 9, 3·014, 6·0638	8, 1, 9, 2·925, 6·0615	10, 1, 12, 3·705, 6·2971	8, 1, 9, 2·912, 6·0615	12, 2, 12, 4·068, 6·3902	10, 1, 18, 4·710, 6·5280	142, 0·73, 0·33, 1510·9, 1272·2, 612·8, 8
10	38, 2, 74, 9·588, 7·8850	70, 2,120, 15·968, 8·4094	40, 3, 68, 9·078, 7·8450	70, 3,120,15·982, 8·4101	44, 5, 62, 8·968, 7·8328	62, 11, 102,13·886, 8·2662	277, 0·58, 0·33, 1680·6, 1280·7, 719·3, 8
11	30, 2, 43, 9·369, 7·4697	24, 2, 14, 4·937, 6·8512	38, 3, 41, 9·995, 7·5501	24, 2, 14, 4·958, 6·8501	30, 4, 43, 9·073, 7·4639	24, 3, 14, 4·937, 6·8519	187, 0·88, 0·50, 2943·1, 2825·5, 1986·6, 8
12	10, 1, 1, 2·129, 5·6971	6, 1, 1, 1.378, 5·2883	8, 1, 1, 1·715, 5·5134	6, 1, 1, 1·378, 5·2933	10, 1, 1, 2·084, 5·7038	6, 1, 1, 1·352, 5·2883	139, 1, 0·66, 3798·0, 3798·0, 3095·1, 8
13	4, 1, 2, 1·210, 5·1299	4, 1, 2, 1·175, 5·1358	4, 1, 2, 1·167, 5·1358	4, 1, 2, 1·182, 5·1385	4, 1, 2, 1·153, 5·1240	4, 1, 2, 1·175, 5·1358	138, 1, 0·66, 5730·3, 5730·3, 5445·9, 8
14	18, 1, 12, 2·937, 6·6120	12, 1, 6, 1·783, 6·1355	26, 3, 11, 3·609, 6·8309	12, 1, 6, 1·794, 6·1356	28, 3, 10, 3·723, 6·8607	12, 1, 6, 1·805, 6·1399	253, 1, 0·66, 4368·8, 4368·8, 3474·6, 8
15	18, 2, 24, 4·969, 7·2738	20, 2, 27, 5·393, 7·3828	18, 2, 26, 5·027, 7·3132	20, 2, 27, 5·414, 7·3840	20, 3, 29, 5·622, 7·4176	20, 3, 24, 5·070, 7·3212	290, 0·90, 0·35, 1892·7, 1829·1, 1227·6, 10
16	32, 3, 37, 10·351, 7·7668	30, 2, 33, 9·162, 7·6756	26, 3, 32, 8·393, 7·5838	30, 3, 33, 9·201, 7·6756	26, 4, 31, 8·300, 7·5684	30, 4, 33, 9·222, 7·6779	228, 0·76, 0·50, 2400·3, 2134·3, 1283·1, 10
17	14, 1, 10, 2·657, 6·7510	20, 2, 15, 3·880, 7·1156	18, 1, 15, 3·625, 7·0475	20, 2, 15, 3·886, 7·1172	14, 1, 10, 2·660, 6·7429	20, 2 15, 3·880, 7·1164	321, 1, 0·66, 4141·5, 4141·5, 2841·8, 10

Seventeen linear fixed charge problems generated by the rules given in pg.223 Using table of random numbers.
(R.L. problems)

* For legend see pg.192

† The problems and the solutions are given in Appendix.B.

Table 9.6

Choice of Problem	LAST IN FIRST OUR SCHEME; Choose Prob=1		RANDOM CHOICE SCHEME; Choose Prob=2		Best (lowest) Lowerbound scheme; Choose prob=3		Problem Constants
Choice of Variable	Random Choice choose var=1	Largest error choice choose var=2	Random Choice choose var=1	Largest error choice choose var=2	Random Choice choose var=1	Largest error choice Choose var=2	
Heuristic Rule No.	1	2	3	4	5	6	
Problem No.	a_1, a_2, a_3, a_4, a_5	a_1, a_2, a_3, a_4, a_5	a_1, a_2, a_3, a_4, a_5,	a_1, a_2, a_3, a_4, a_5	a_1, a_2, a_3, a_4, a_5	a_1, a_2, a_3, a_4, a_5	pr, qr, sr, Zp, Zopt, ZQ, V
1	10, 2, 14, 2·505, 5·5760	8, 1, 8, 1·781, 5·2418	10, 1, 20, 3·029, 5·7589	8, 1, 8, 1·808, 5·2470	10, 1, 20, 3·038, 5·7589	8, 1, 8, 1·781, 5·2364	105, 1·00, 0·33, 1755·9, 1755·9, 1307·0, 5,
2	6, 1, 8, 1·533, 5·0876	6, 1, 6, 1·394, 4·9904	8, 1, 10, 1·933, 5·3230	6, 1, 6, 1·404, 4·9904	6, 1, 8, 1·533, 5·0876	6, 1, 6, 1·422, 4·9904	102, 0·64, 0·33, 1733·9, 1450·6, 957·7, 5,
3	6, 1, 4, 1·041, 4·8598	4, 1, 2, 0·713, 4·4886	6, 1, 4, 1·049, 4·8598	4, 1, 2, 0·721, 4·4886	6, 1, 4, 1·024, 4·8520	4, 1, 2, 0·707, 4·4886	122, 1·00, 0·50, 2837·9, 2837·9, 2296·1, 5,
4	4, 1, 6, 1·739, 4·8040	4, 1, 6, 1·765, 4·8040	4, 1, 6, 1·779, 4·8040	4, 1, 6, 1·754, 4·8122	4, 1, 6, 1·779, 4·8040	4, 1, 6, 1·754, 4·8122	69, 1·00, 0·50, 1355·1, 1355·1, 1002·1, 5,
5	6, 1, 17, 2·311, 5·4806	8, 1, 22, 3·020, 5·7366	10, 1, 27, 3·650, 5·9323	8, 1, 22, 2·962, 5·7366	8, 1, 24, 3·155, 5·7869	8, 1, 22, 2·990, 5·7366	103, 1·00, 0·50, 1525·5, 1525·5, 986·2, 5,
6	2, 1, 4, 0·929, 4·3820	2, 1, 4, 0·918, 4·3820	4, 1, 10, 1·812, 5·0499	2, 1, 4, 0·929, 4·3820	4, 1, 10, 1·812, 5·0499	2, 1, 4, 0·940, 4·3945	84, 1·00, 0·60, 871·1, 871·1, 731·3, 5,
7	10, 1, 18, 3·482, 5·6971	8, 1, 12, 2·593, 5·4181	6, 1, 12, 2·279, 5·2832	8, 1, 12, 2·624, 5·4161	6, 1, 12, 2·029, 5·2933	8, 1, 12, 2·593, 5·4117	86, 1·00, 0·60, 5063·5, 5063·5, 4446·7, 5,
8	12, 1, 53, 5·209, 7·1884	8, 1, 16, 2·067, 6·2653	8, 1, 16, 2·071, 6·2635	8, 1, 16, 2·067, 6·2653	10, 1, 41, 4·119, 6·9508	8, 1, 16, 2·083, 6·2653	252, 0·42, 0·33, 1523·8, 1162·4, 894·9, 8,
9	12, 2, 34, 4·592, 6·8773	10, 2, 25, 3·509, 6·6147	14, 2, 33, 4·720, 6·9058	10, 2, 25, 3·514, 6·6147	10, 2, 25, 3·531, 6·6161	10, 2, 25, 3·531, 6·6147	211, 0·82, 0·33, 1557·9, 1393·3, 659·8, 8,
10	30, 2,535, 27·596, 9·2737	20, 2, 41, 3·408, 7·1808	22, 1, 224, 14·262, 8·6112	20, 2, 41, 3·399, 7·1808	32, 4, 102, 7·163, 7·2950	20, 3, 41, 3·375, 7·1816	387, 0·50, 0·33, 2023·2, 1400·6, 773·9, 8,
11	20, 1, 52, 3·927, 7·3146	8, 1, 18, 1·473, 6·3386	14, 1, 36, 2·780, 6·9698	8, 1, 18, 1·473, 6·3386	14, 1, 36, 2·773, 6·9698	8, 1, 18, 1·473, 6·3386	383, 0·99, 0·50, 3057·2, 3056·3, 2099·4, 8,
12	10, 1, 1, 1·278, 5·6870	8, 1, 1, 1·061, 5·094	12, 1, 2, 1·576, 5·8944	8, 1, 1, 1·065, 5·5094	10, 1, 2, 1·358, 5·7462	8, 1, 1, 1·065, 5·5094	230, 0·68, 0·66, 4311·5, 3976·5, 3268·8, 8,
13	6, 1, 4, 0·851, 5·5134	6, 1, 4, 0·858, 5·5174	4, 1, 2, 0·564, 5·0999	6, 1, 4, 0·862, 5·5215	4, 1, 2, 0·566, 5·1060	6, 1, 4, 0·858, 5·5215	289, 0·66, 0·66, 6355·3, 6203·8, 5914·7, 8,
14	14, 1, 12, 1·486, 6·4069	10, 1, 10, 1·140, 6·1463	14, 1, 14, 1·580, 6·4677	10, 1, 10, 1·147, 6·1485	10, 1, 10, 1·140, 6·1442	10, 1, 10, 1·147, 6·1506	407, 1·00, 0·66, 5119·5, 5119·5, 4225·4, 8,
15	16, 2, 59, 3·843, 7·7319	16, 2, 49, 3·364, 7·6004	13, 2, 362, 17·559, 9·2526	16, 2, 49, 3·373, 7·6019	24, 3, 75, 5·066, 8·0083	16, 2, 49, 3·369, 7·6019	594, 0·66, 0·35, 2338·8, 1989, 1313·2, 10,
16	24, 3, 58, 6·785, 7·8237	36, 3, 84, 9·981, 8·2066	32, 4, 84, 9·606, 8·1708	30, 3, 71, 8·445, 8·0369	32, 4, 72, 8·669, 8·0361	30, 4, 71, 8·431, 8·0375	367, 0·78, 0·50, 2458, 2216, 1341, 10,
17	14, 1, 21, 2·060, 7·0103	12, 1, 17, 1·729, 6·8373	18, 1, 38, 3·221, 7·4582	12, 1, 17, 1·725, 6·8373	16, 1, 26, 2·454, 7·1869	12, 1, 17, 1·732, 6·8395	538, 1·00, 0·66, 4246, 4246, 2947, 10,

Seventeen Quadratic Fixed Charge Problems generated by rules given in Pg.329
Using table of random numbers.

(R.Q. Problems)

* For legend see Pg.192

† The problems and the solutions are given in Appendix.B.

Production Scheduling Problems (P.S.P.)

Heuristic Rule No.	No. of variables/ constraints	No. of observations	Mean (ln (comp. instructions)) [std dev] μ_{a_6} [σ_{a_6}]	Mean (ln (Max. store used)) [std dev] $\mu(\ln a_3)$ [$\sigma(\ln a_3)$]
1	5/5	2	5·736[0·14]	0·693[0·00]
	6/6	2	6·225[0·11]	0·896[0·20]
	7/7	2	7·639[0·04]	1·498[0·11]
	8/8	2	8·150[0·05]	1·386[0·00]
	9/9	2	9·079[0·10]	1·869[0·08]
2	5/5	2	5·106[0·08]	0·549[0·55]
	6/6	2	6·225[0·11]	0·693[0·00]
	7/7	2	7·317[0·03]	1·243[0·14]
	8/8	2	8·233[0·02]	1·386[0·00]
	9/9	2	8·610[0·20]	1·498[0·11]
3	5/5	2	5·676[0·16]	0·693[0·00]
	6/6	2	7·495[0·49]	1·242[0·14]
	7/7	2	7·803[0·18]	1·903[0·29]
	8/8	2	8·237[0·04]	1·896[0·08]
	9/9	2	8·929[0·15]	2·565[0·00]
4	5/5	2	5·100[0·08]	0·000[0·00]
	6/6	2	5·938[0·00]	0·693[0·00]
	7/7	2	7·291[0·00]	1·498[0·11]
	8/8	2	8·251[0·09]	1·844[0·24]
	9/9	2	8·546[0·21]	1·869[0·08]

Table 9·7 (continued on next page)

Production Scheduling Problems (P.S.P.)
(continued from the previous page).

Heuristic Rule No.	No. of variables/ constraints	No. of observa-tions	Mean (ln (comp. instructions)) [std dev] μ_{a_5} [σ_{a_5}]	Mean (ln (Max. store used)) [std dev] $\mu(\ln a_2)$ [$\sigma(\ln a_2)$]
5	5/5	2	5·563[0·27]	0·549[0·55]
	6/6	2	6·794[0·14]	1·935[0·14]
	7/7	2	7·566[0·06]	2·394[0·09]
	8/8	2	8·521[0·42]	2·776[0·07]
	9/9	2	8·893[0.19]	2·637[0·07]
6	5/5	2	5·103[0·07]	0·000[0·00]
	6/6	2	5·939[0·07]	0·693[0·00]
	7/7	2	7·240[0·00]	2·197[0·00]
	8/8	2	8·190[0·06]	2·708[0·00]
	9/9	2	8·562[0·10]	2·565[0·00]

Table 4·7 (continued)

Random Linear Problems (R.L.P.)

Heuristic Rule No.	No. of variables/ constraints	No. of observations	Mean (ln(comp. instructions)) [std dev] μ_{a_5} [σ_{a_5}]	Mean (ln (max store used)) [std dev] $\mu(\ln a_3)$ [$\sigma \ln(a_3)$)]
1	5/5	7	5·079[0·35]	0·099[0·25]
	8/8	7	6·490[0·90]	0·294[0·34]
	10/10	3	7·263[0·03]	0·597[0·45]
2	5/5	7	5·044[0·34]	0·00[0·00]
	8/8	7	6·269[1·02]	0·183[0·32]
	10/10	3	7·391[0·07]	0·693[0]
3	5/5	7	5·079[0·35]	0·00[0·00]
	8/8	7	6·452[0·95]	0·671[0·25]
	10/10	3	7·315[0·23]	0·597[0·45]
4	5/5	7	5·049[0·33]	0·00[0·00]
	8/8	7	6·270[1·02]	0·256[0·42]
	10/10	3	7·392[0·22]	0·828[0·19]
5	5/5	7	5·064[0·34]	0·099[0·25]
	8/8	7	6·603[0·46]	0·841[0·59]
	10/10	3	7·243[0·37]	0·828[0·60]
6	5/5	7	5·048[0·30]	0·000[0·00]
	8/8	7	5·991[0·61]	0·183[0·37]
	10/10	3	7·371[0·23]	1·059[0·28]

Table 9·8

Random Quadratic Problems (R.Q.P.)

Heuristic Rule No.	No. of variables/ constraints	No. of Observations	Mean (ln (comp. instructions)) [std dev] μ_{a_3} [σ_{a_3}]	Mean (ln (Max store used)) [std dev] $\mu(\ln a_3)$ [$\sigma(\ln a_3)$]
1	5/5	7	5·127[0·44]	0·099[0·25]
	8/8	7	6·894[0·88]	0·159[0·33]
	10/10	3	7·522[0·36]	0·597[0·45]
2	5/5	7	5·009[0·46]	0·000[0·00]
	8/8	7	6·225[0·63]	0·183[0·32]
	10/10	3	7·548[1·00]	0·597[0·45]
3	5/5	7	5·287[0·40]	0·000[0·00]
	8/8	7	6·602[1·01]	0·099[0·25]
	10/10	3	8·294[0·73]	0·693[0·57]
4	5/5	7	5·010[0·33]	0·000[0·00]
	8/8	7	6·226[0·54]	0·183[0·32]
	10/10	3	7·492[0·52]	0·597[0·45]
5	5/5	7	5·233[0·37]	0·000[0·00]
	8/8	7	6·494[1·15]	0·297[0·51]
	10/10	3	7·744[0·39]	0·828[0·59]
6	5/5	7	5·011[0·46]	0·000[0·00]
	8/8	7	6·226[0·53]	0·256[0·42]
	10/10	3	7·493[0·49]	0·693[0·57]

Table 9·9

Production Scheduling Problems

Heuristic Rule-2	MATRIX OF CORRELATION COEFFICIENTS			
Sample Size=10	1 ln (max store used)	2 ln (number of computing instructions)	3 No. of variables/ constraints	4 Sparsity
1 ln (max store)	r_{11} =1	Coefficient of partial correlation $r_{12\cdot3}=\frac{r_{21}-r_{31}r_{32}}{\sqrt{(1-r_{31}^2)(1-r_{32}^2)}}=0.826$		Constant=0·50 (not considered)
2 ln (Comp.instr.)	r_{21} =0·960	r_{22} =1		
3 no. of variables /constraints	r_{31} =0·929	r_{32} =0·982	r_{33}=1	
4 Sparsity	Constant=0·50 (not considered)			

Two variable Regression Parameters	Regression of 1 on 3	Regression of 2 on 3
Estimate of the slope coeff [std error]	0·3689[0·0519]	0·9302[0·0629]
Estimate of the intercept [std error]	-1·61183[0·0505]	0·5292[0·4429]

Heuristic Rule-6	MATRIX OF CORRELATION COEFFICIENTS			
Sample Size=10	1	2	3	4
1	r_{11} =1	Coefficient of partial correlation $r_{12\cdot3}=\frac{r_{21}-r_{31}r_{32}}{\sqrt{(1-r_{31}^2)(1-r_{32}^2)}}=0.890$		Constant=0·50 (not considered)
2	r_{21} =0·974	r_{22}=1		
3	r_{31} =0·931	r_{32} =0·984	r_{33}=1	
4	constant=0·50 (not considered)			

Two variable regression parameters	Regression of 1 on 3	Regression of 2 on 3
Estimate of the slope coeff [std error]	0·7145[0·0992]	0·9169[0·0586]
Estimate of the intercept [std error]	-3·3687[0·1624]	0·5886[0·4104]

Table 9·10

Random Linear Problems

Heuristic Rule-2	MATRIX OF CORRELATION COEFFICIENTS			
Sample Size=17	1 ln (max store used)	2 ln (number of comp. instr.)	3 No. of variables/ constraints	4 Sparsity
1 ln (max store used)	γ_{11} =1	Coefficient of partial correlation $r_{12\cdot3}=\frac{r_{21}-r_{31}r_{32}}{\sqrt{(1-r_{31}^2)(1-r_{32}^2)}}$ = 0·767		
2 ln (no. of comp. instr.)	γ_{21} =0·890	γ_{22} =1		
3 No. of variables /constraints	γ_{31} =0·705	γ_{32} =0·777	γ_{33} =1	
4 Sparsity	γ_{41} =0·112	γ_{42} =0·252	γ_{43}=0·071	γ_{44} =1

Two variable regression parameters	Regression of 1 on 3	Regression of 2 on 3
Estimate of the slope coeff [std error]	0·1168[0·0303]	0·4510[0·0943]
Estimate of the intercept [std error]	-0·6278[0·007]	2·7518[0·5623]

Heuristic Rule-6	MATRIX OF CORRELATION COEFFICIENTS			
Sample Size=16 †	1	2	3	4
1	γ_{11} =1	Coefficient of partial correlation $r_{12\cdot3}=\frac{r_{21}-r_{31}r_{32}}{\sqrt{(1-r_{31}^2)(1-r_{32}^2)}}$ =0·750		
2	γ_{21} =0·868	γ_{22} =1		
3	γ_{31} =0·693	γ_{32} =0·863	γ_{33} =1	
4	γ_{41} =0·249	γ_{42} =0·100	γ_{43}=0·113	γ_{44} =1

Two variable regression parameters	Regression of 1 on 3	Regression of 2 on 3
Estimate of the slope coeff [std error]	0·1701[0·0473]	0·4173[0·0654]
Estimate of the intercept [std error]	-0·9340[0·0131]	2·8668[0·3801]

† Observation 10 excluded (cf table 10·5)

Table 9·11.

Random Quadratic Problems

Heuristic Rule-2	MATRIX OF CORRELATION COEFFICIENTS			
Sample Size=17	1 ln (max store used)	2 ln (number of comp. instr.)	3 No. of variables /constraints	4 Sparsity
1 ln (max store used)	r_{11} =1	Coefficient of partial correlation $r_{12\cdot3} = \frac{r_{21} - r_{31} r_{32}}{\sqrt{(1-r_{31}^2)(1-r_{32}^2)}}$ =0·723		
2 ln (no. of comp instr.)	r_{21} =0·789	r_{22} =1		
3 No. of variables /constraints	r_{31} =0·567	r_{32} =0·863	r_{33} =1	
Sparsity	r_{41} =0·409	r_{42} =0·268	r_{43} =0·004	r_{44} =1

Two variable regression parameters	Regression of 1 on 3	Regression of 2 on 3
Estimate of the slope coeff [std error]	0·1034[0·0388]	0·4800[0·0725]
Estimate of the intercept [std error]	-0·5492[0·0078]	2·5448[0·4318]

Heuristic Rule-6	MATRIX OF CORRELATION COEFFICIENTS			
Sample Size=17	1	2	3	4
1	r_{11} =1	Coefficient of partial correlation $r_{12\cdot3} = \frac{r_{21} - r_{31} r_{32}}{\sqrt{(1-r_{31}^2)(1-r_{32}^2)}}$ =0·719		
2	r_{21} =0·768	r_{22} =1		
3	r_{31} =0·538	r_{32} =0·866	r_{33} =1	
4	r_{41} =0·387	r_{42} =0·230	r_{43} =0·064	r_{44} =1

Two variable regression parameters	Regression of 1 on 3	Regression of 2 on 3
Estimate of the slope coeff [std error]	0·1226[0·0496]	0·4692[0·0699]
Estimate of the intercept [std error]	-0·6452[0·0119]	2·6093[0·4157]

Table 9·12

Regression of ln (computing instructions)

+ln (max store used) on no. of variables

Problem Type	Sample Size	Heuristic Rule-2		Heuristic Rule-6	
		Estimate of the slope [std error]	Estimate of the intercept [std error]	Estimate of the slope [std error]	Estimate of the intercept [std error]
Production Scheduling Problems	10	1·2991 [0·1053]	-1·0891 [0·8431]	1·6314 [0·1538]	-2·7800 [1·3285]
Random Linear Problems	17	0·5679 [0·1191]	2·1240 [0·7347]	† 0·5874 [0·1054]	† 1·9327 [0·6408]
Random Quadratic Problems	17	0·5834 [0·1040]	1·9956 [0·6392]	0·5918 [0·1110]	1·9641 [0·6854]

† Sample size=16

Table 9·13

	Number of Structural variables	Number of Upper bound constraint	Number of Iterations	Total Number of Computing Instructions
Integer Q. Prog. Formulation	10	10	11	1380
Formulation by the approx. technique	5	5	11	346

Table 9·15

Problem Type	No. of observations	Mean and std-deviation. $\bar{Z}$; S_Z	t-coefficient $t_{n-1} = \frac{\bar{Z}}{\sqrt{\frac{1}{n}(S_Z)^2}}$	Significance Level
Production Scheduling	10	$\bar{Z}_P = 0{\cdot}634$ $S_{ZP} = 0{\cdot}604$	$t_9 = 3{\cdot}392$	> 1% level
Random Quadratic	17	$\bar{Z}_Q = 0{\cdot}033$ $S_{ZQ} = 0{\cdot}086$	$t_{16} = 1{\cdot}587$	< 5% level
Random Linear	16	$\bar{Z}_L = 0{\cdot}123$ $S_{ZL} = 0{\cdot}215$	$t_{15} = 1{\cdot}5865$	≃ 5% level

Table 9·14

Legend

Choice of variables

Choose var=1
The variable is
chosen randomly

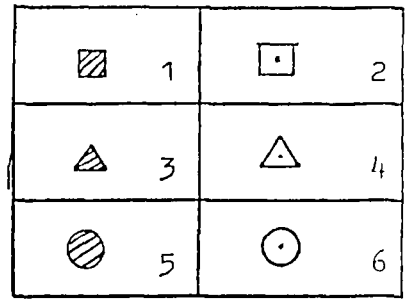
Choose var=2
The variable is
chosen by the largest
error reduction scheme.

Choice of Problem

Last-in-First Out (choose problem=1)

Randomly chosen (choose problem=2)

Problem with lowest lower bound chosen
(choose problem=3)



Heuristic Rules and Symbols.

Fig. 9.5a.

GM:SA
3 April 1968

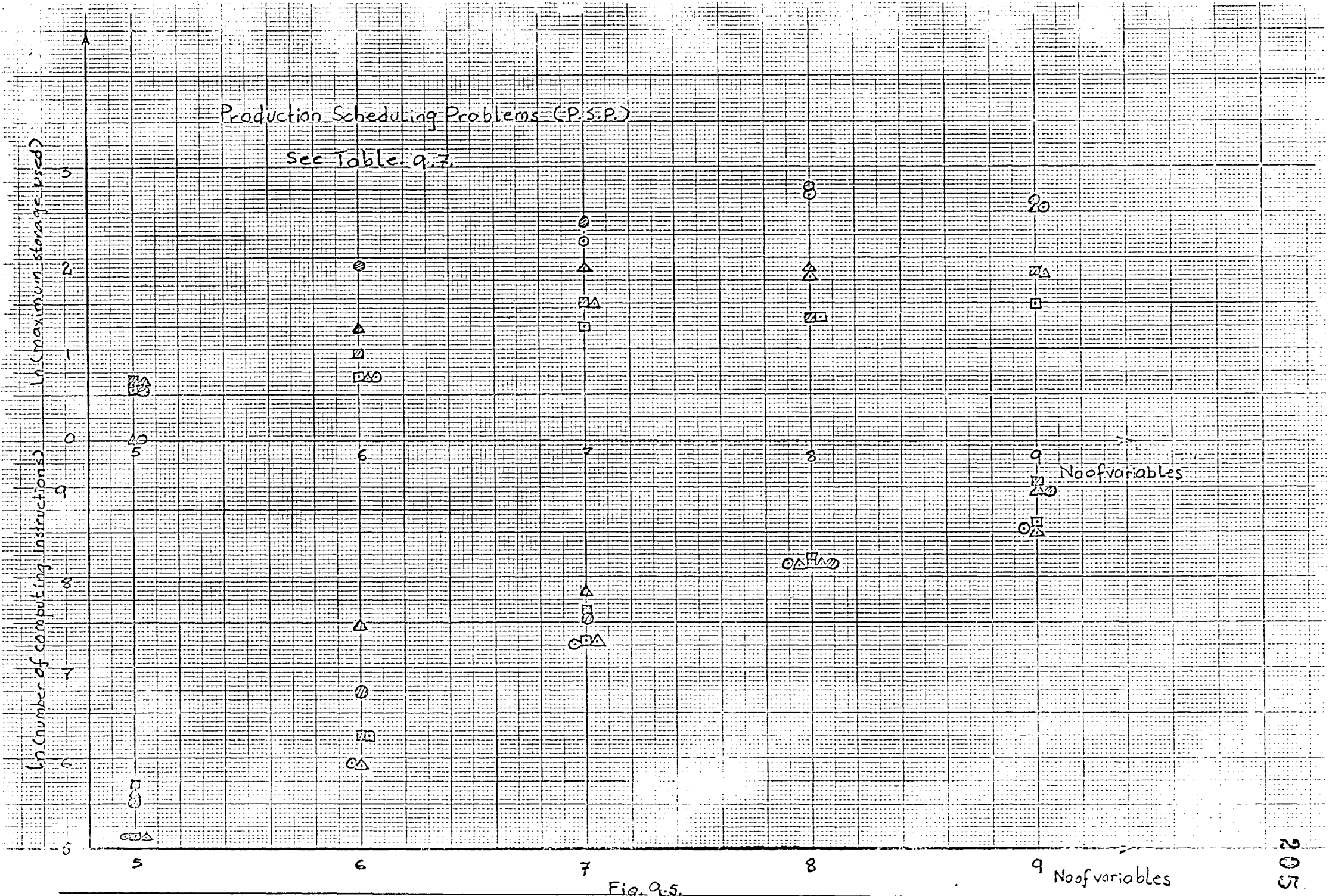

Production Scheduling Problems (P.S.P.)
See Table 9.7.
Ln (maximum storage used)
Ln (number of computing instructions)
No of variables
No of variables
5
6
7
8
9
1
2
3
4
5
6
7
8
9

Fig. 9.5.

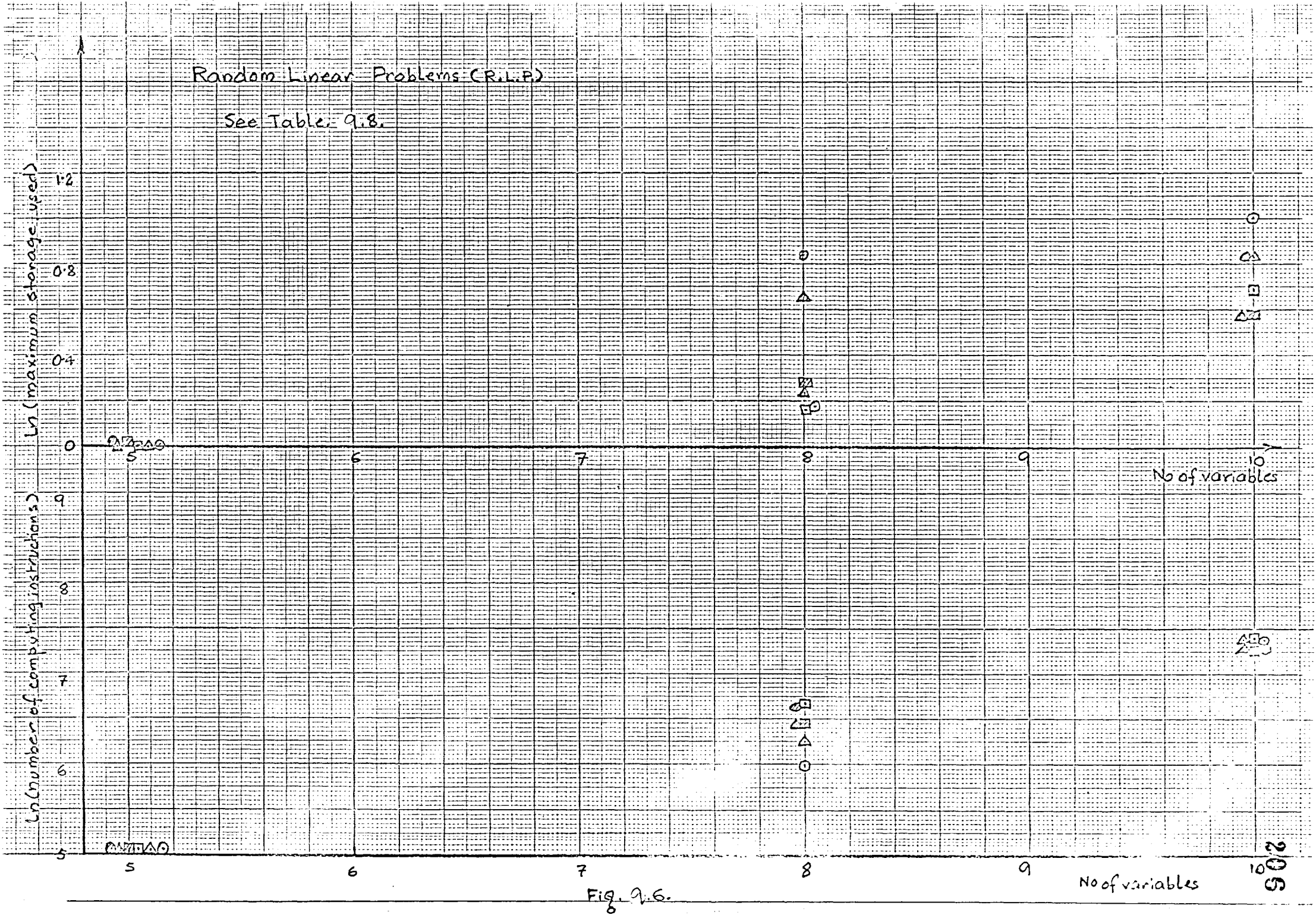

Random Linear Problems (R.L.P.)
See Table: 9.8.
Ln (maximum storage used)
Ln (number of computing instructions)
No of variables
No of variables
1·2
0·8
0·4
0
9
8
7
6
5
5
6
7
8
9
10

Fig. 9.6.

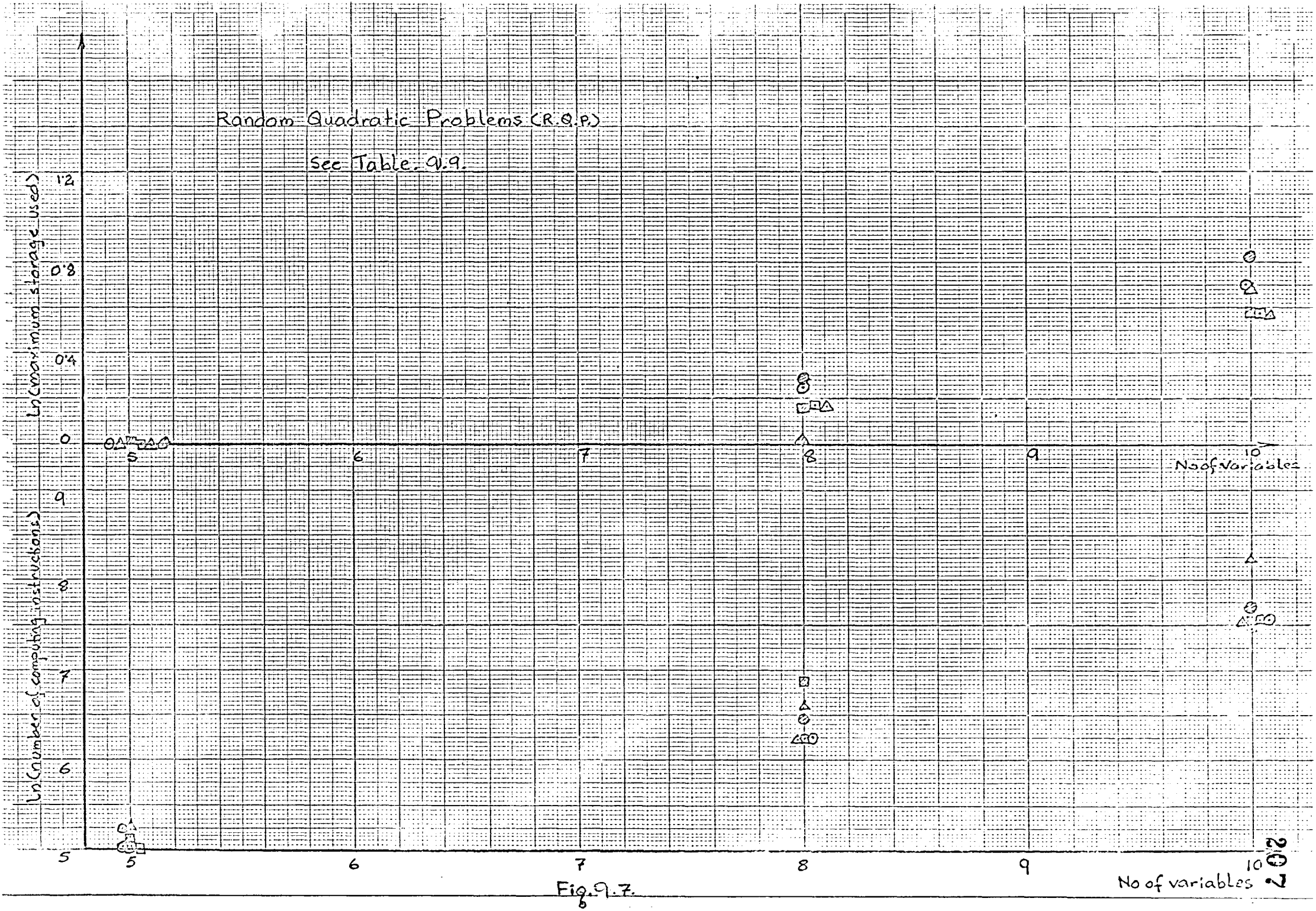

Random Quadratic Problems (R.Q.P.)
See Table 9.9.
ln(maximum storage used)
ln(Number of computing instructions)
No of variables
No of variables
1·2
0·8
0·4
0
9
8
7
6
5
5
6
7
8
9
10

Fig. 9.7.

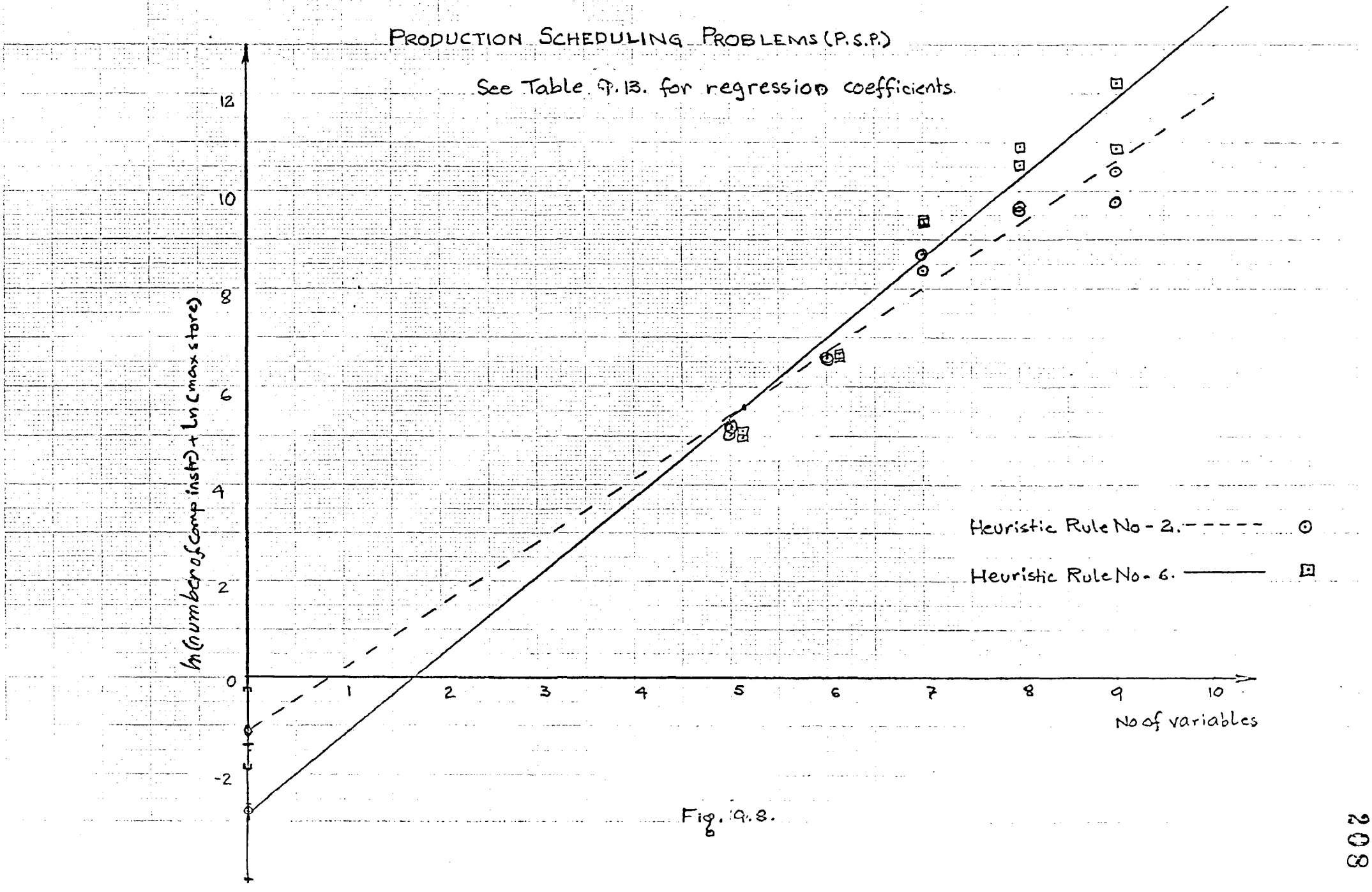

PRODUCTION SCHEDULING PROBLEMS (P.S.P.)
See Table 9.13. for regression coefficients.
ln (number of comp instr) + ln (max store)
12
10
8
6
4
2
0
-2
1
2
3
4
5
6
7
8
9
10
No of variables
Heuristic Rule No - 2.
Heuristic Rule No - 6.

Fig. 9.8.

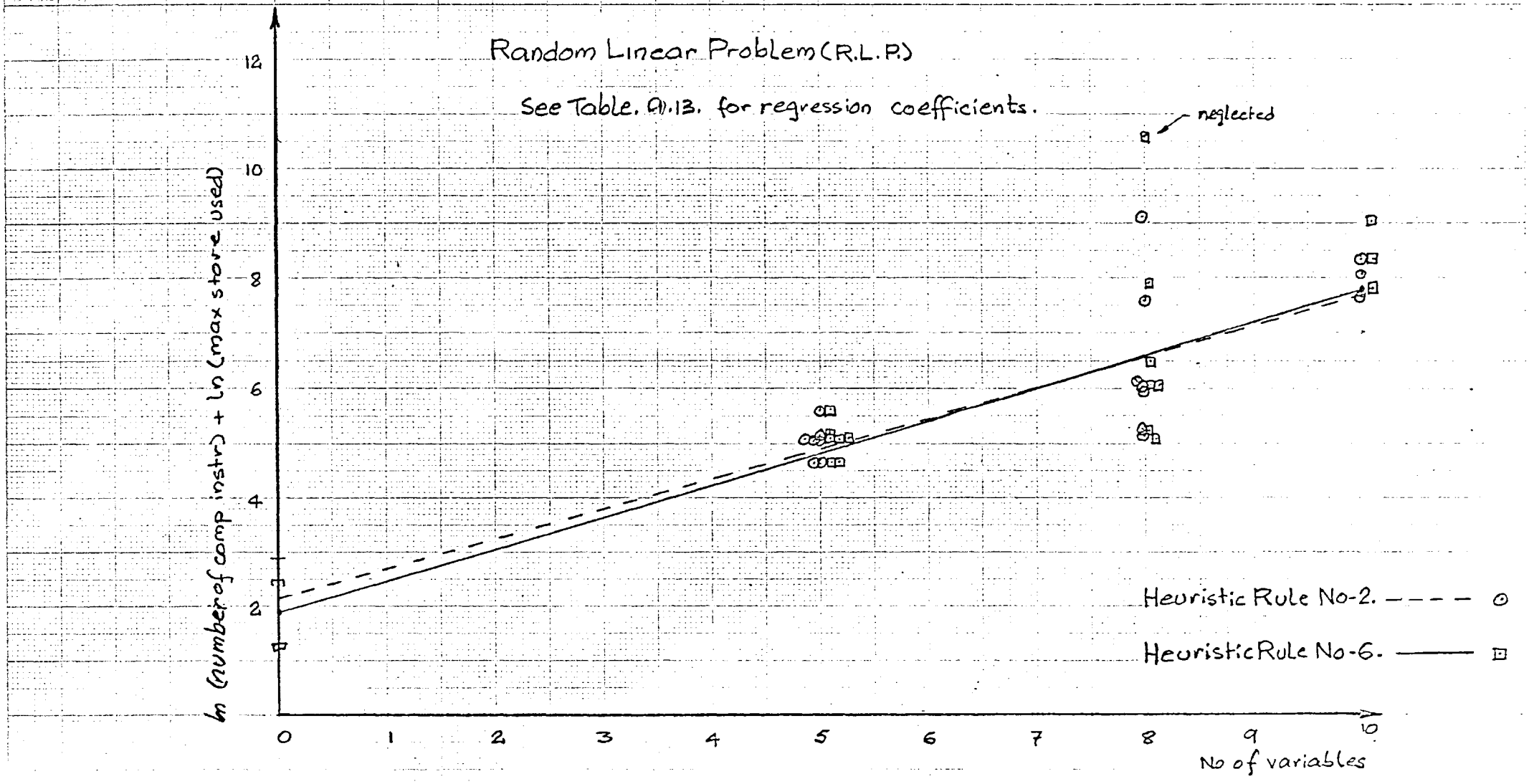

Random Linear Problem (R.L.P.)
See Table. 9.13. for regression coefficients.
neglected
ln (number of comp instr) + ln (max store used)
12
10
8
6
4
2
0
1
2
3
4
5
6
7
8
9
10
No of variables
Heuristic Rule No-2.
Heuristic Rule No-6.

Fig. 9.9.

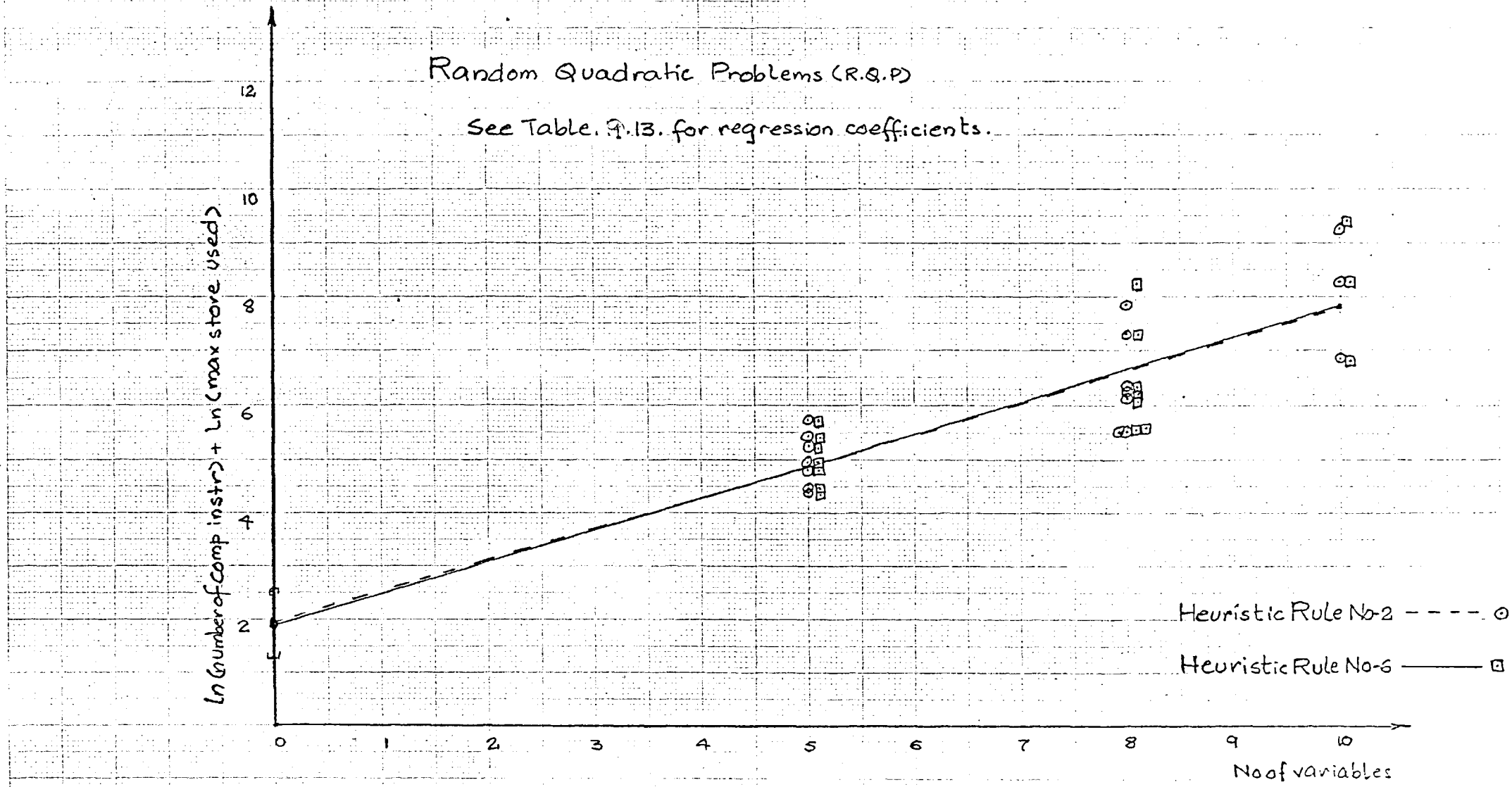

Random Quadratic Problems (R.Q.P)
See Table. 9.13. for regression coefficients.
Ln (Number of Comp instr) + Ln (max store used)
12
10
8
6
4
2
0
1
2
3
4
5
6
7
8
9
10
No of variables
Heuristic Rule No-2
Heuristic Rule No-5

Fig. 9.10.

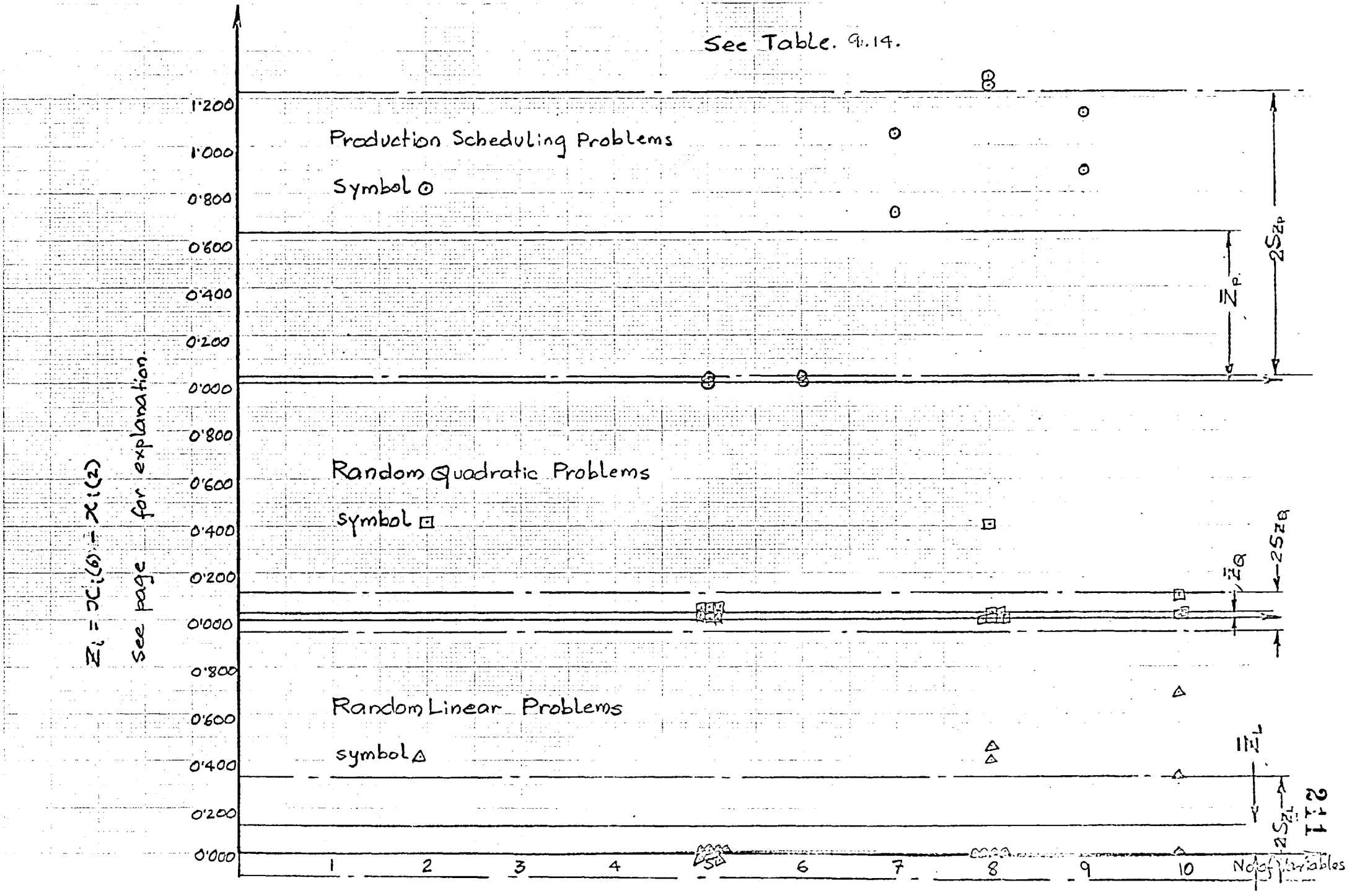
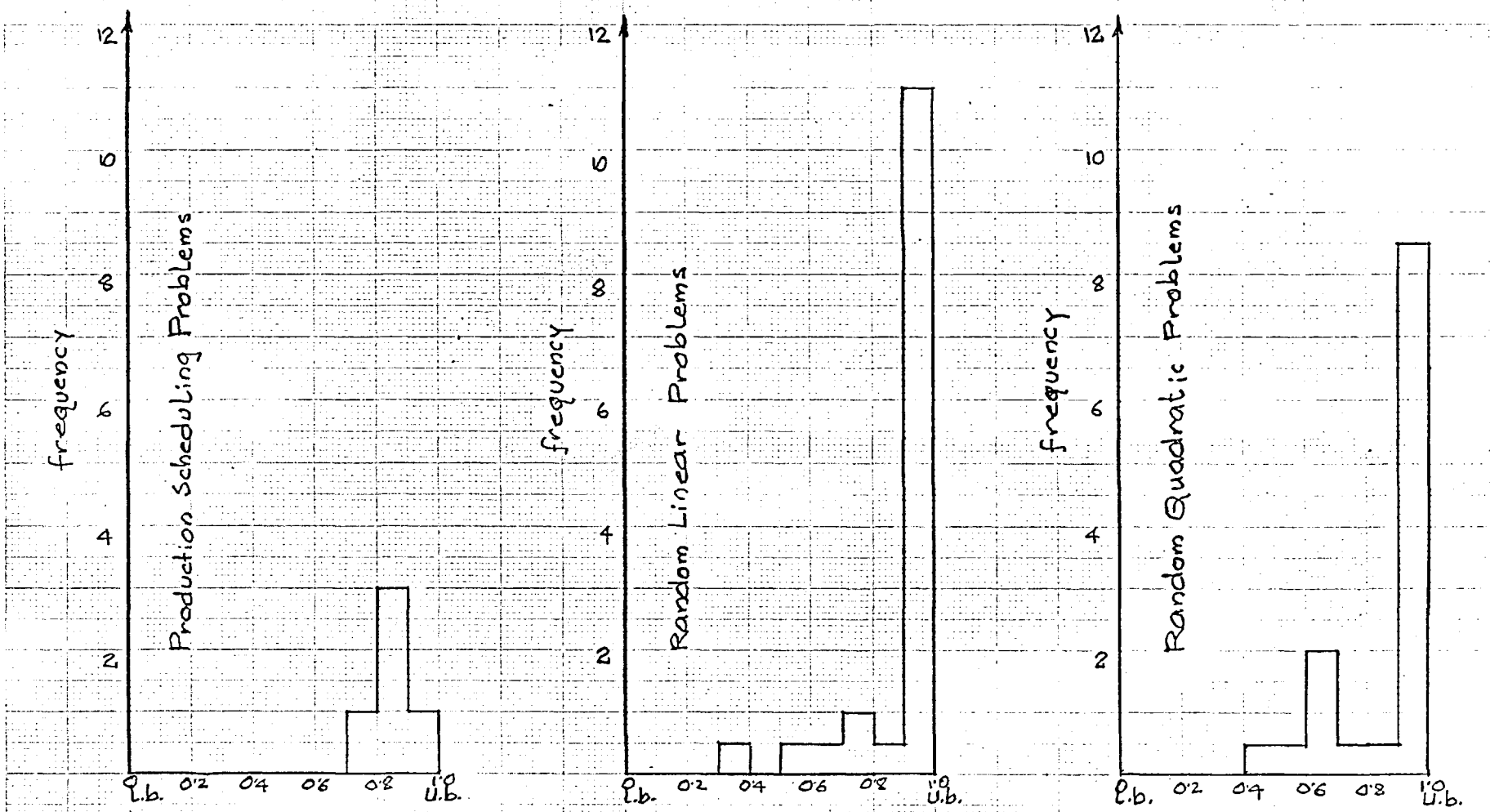

See Table. 9.14.
Production Scheduling Problems
Symbol ⊙
Random Quadratic Problems
symbol ⊡
Random Linear Problems
symbol △
$Z_i = x_i(6) - x_i(2)$
see page for explanation.
1·200
1·000
0·800
0·600
0·400
0·200
0·000
0·800
0·600
0·400
0·200
0·000
0·800
0·600
0·400
0·200
0·000
$\bar{Z}_P$
$2S_{Z_P}$
$\bar{Z}_Q$
$2S_{Z_Q}$
$\bar{Z}_L$
$2S_{Z_L}$
2·11
1 2 3 4 5 6 7 8 9 10
No. of variables

Fig. 9.11.



Distribution of the optimal solution between inital upper and lower bounds.

Fig. 7.12.

PART 3

References

3.1 Land, A.H., and Doig, A.G., An Automatic Method of Solving Discrete Programming Problems, Econometrica, 28 (1960), 497-520.

3.2 Little, John, D.C., Murty, Katta, G., Sweeney, Dura, N., and Karel, Caroline, An Algorithm for the Travelling Salesman Problem, Opns Res, (1963), 972-989.

3.3 Dakin, R.J., A Tree-Search Algorithm for Mixed Integer Programming Problems, Computer Journal, 8 (1965), 250-255.

3.4. Beale, E.M.L., and Small, R.E., Mixed Integer Programming by a Branch and Bound Technique, Proceedings IFIP Congress 2 (1965).

3.5 Driebeck, N.J., An Algorithm for the Solution of Mixed Integer Programming Problems, Management Science, 12 (1966), 576-587.

3.6 Burstall, R.M., A Tree Searching Method for Solving Integer Linear Inequalities, University of Edinburgh, Experimental Programming Unit, Reports No. 10, October (1965).

3.7 Kunzi, H.P., Oettli, W., Integer Quadratic Programming, Recent Advances in Mathematical Programming, Ed. Graves R.L., and Wolfe, P., McGrawHill, (1963) 303-308.

3.8 Kuhn, H.W. and Baumol, W.J., Algorithm for Fixed Charges Problem, Naval Research Logistics Quarterly, 9 (1962), 1-15.

3.9 Balinski, M.L., Fixed Cost Transportation Problems, Naval Research Logistcs Quarterly, 8 (1961), 2, 41-54.

3.10 Dwyer, P.S., Use of Completely Reduced Matrices in Solving Transportation Problem with Fixed Charges, Naval Research Logistics Quarterly, 73 (1966), 3, 289-314.

3.11 Cooper, L., and Drebes, C., An Approximate Solution Method for the Fixed Charge Problem, Naval Research and Logistics Quarterly, 14 (1967), 1, 101-114.

3.12 Abadie, J., Non-linear Programming, North Holland Publishing Co., - Amsterdam (1967), 173-188.

3.13 Hadley, G., and Whitin, T.M., Analysis of Inventory Systems, Prentice Hall, (1963), 323-345.

3.14 Moses, L.E., Oakford, R.V., Tables of Random Permutations, London. George Allen and Unwin Ltd., (1962).

3.15 Bayer, G., Transportation Problem, Algorithm 293, Communications of the A.C.M., 9 (1966), 12, 869-871.

3.16 Bellman, R., On the Computational Solution of Linear Programming Problems Involving Almost Block Diagonal Matrices, Management Science, 3 (1957).

3.17 Nemhauser, G.L., Decomposition of Linear Programs by Dynamic Programming, Naval Research Logistics Quarterly, 11, (1964), 191-195.

3.18 Gilmore, P.C., and Gomory, R.E., A Linear Programming Approach to the Cutting Stock Problem, Operations Research, 9, (1961), 849-859.

3.19 - and - , A Linear Programming Approach to the Cutting Stock Problem, Operations Research, 11, (1963), 863-888.

3.20 - and - , Multi-stage Cutting Stock Problems of Two and More Dimensions, Operations Research, 13, 94-120.

3.21 - and - , The Theory and Computation of Knapsack Functions, Operations Research, 14 (1966), 1054-1074.

3.22 Shapiro, J.F., Dynamic Programming Algorithms for the Integer Programming Problem-I. The Integer Programming Problem Viewed as a Knapsack Type Problem, Operations Research, 16, (1968), 103-121.

3.23 Reiter, S., and Rice, D.B., Discrete Optimizing Solution Procedures for Linear and Nonlinear Integer Programming Problems, Management Science, 12, (1966), (829-840).

3.24 Burstall, R.M., Tree-Searching Methods with an Application to a Network Design Problem, Machine Intelligence 1, ed. Collins, N.L., and Michie, D., Oliver and Boyd, (1967), 65-85.

3.25 Michie, D., Game-Playing and Game-Learning Automata, Advances in Programming and Non-numerical Computation, Pergamon Press (1966), 183-200.

3.26 Falk, J.E., and Soland, R.M., An Algorithm for Separable Nonconvex Programming, paper presented at SIAM conference on Optimization June 11-14, 1968. Toronto, Canada.

3.27 Lawler, E.W., Wood, D.E., Branch-and-Bound Methods: A Survey, Operations Research, 14, No. 5 (1966). 699-719.

3.28 Demianov, V.F., and Khudiakov, L. Iu., Solution of an Integer Problem of Quadratic Programming, PMM (Russian-Trans/English). (1965), 170-174.

3.29 Wagner, H.M., and Whitin, T.M., A Dynamic Version of the Economic Lot Size Model, Management Science, 5, (1958), 89-96.

Chapter 10

Concluding Remarks and Discussion

Introduction

In this chapter some important features of the problems discussed in the thesis are reconsidered. The views presented are an outgrowth of the conclusions reached in the different parts of the thesis, and an appraisal of the related works of other investigators in the field.

In Section a the incorporation of dynamic programming concepts within the (specifically linear) mathematical programming computational scheme is considered. In Section b the relevance of tree search methods in problem-solving and combinatorial optimization method is discussed. In Section c the question of what defines a heuristic method is posed and an attempt is made to answer this. Section d outlines the importance of utilizing the structure of the model for which one is trying to develop a solution technique.

a. Dynamic programming concepts in mathematical programming techniques

The decomposition by dynamic programming analysis of linear programs with a slightly intertwined constraint matrix was first suggested by Bellman [3.16], and more recently Nemhauser [3.17] has also investigated this approach. It seems that the highly structured quadratic program considered in Chapter 6 is amenable to similar analysis. The saving in computing time and storage that should result from such an approach will allow a longer section of a road to be considered at any one time; this could be a profitable branch of study.

The other area of application of dynamic programming techniques within linear programs lies in the solution of integer linear programs. Gilmore and Gomory in a series of papers([3.18], [3.19], [3.20]) considered a very special integer programming problem, namely the cutting stock problem. The theory developed in these papers culminated in another paper due to

Gomory [2.8], the result of which may be summarized as follows:

From the solution of the continuous linear programming problem one can obtain the optimal integer solution to the integer programming problem by solving an associated knapsack problem.

The efficient computation of the knapsack problem has again been considered by Gilmore and Gomory [3.21]. Shapiro [3.22] has investigated the same approach recently and Balinski [3.12] has described a similar solution technique which he calls the rounding algorithm. The work reported in Chapter 4 bears a direct relationship to the works just quoted. However, the auxiliary problem which requires to be solved by this approach is not a knapsack but a cargoloading problem. In this method, the reduced cost coefficients in the continuous optimum solution are used to define a hyperplane parallel to the objective. The location of the relevant lattice points in this plane may be achieved by dynamic programming, following the formulation of Finkelshty'n [2.10]. The alternative approach of solving diophantine equations in nonnegative integers, as shown in Chapter 4, may be claimed to be more economical from the viewpoint of storage used, but naturally at the expense of increased computing time. Unfortunately, computing experience with these methods, both that of the author and these reported by other workers, seems to be very meagre and hardly justifies definite conclusions about the usefulness of this approach as a general purpose integer programming method. Clearly more computing work needs to be done in this area.

b. Tree search methods their areas of application and use.

The applications of the tree search algorithms considered in Part 3 of the thesis appear purely in the context of mathematical programming. In the field of artificial intelligence([1.23],[3.24],[3.25]) also, there are occasions where it is necessary to devise similar tree search schemes. The travelling salesman problem([3.27]) can be considered to belong to both these areas of study. Developments in solution techniques in these two fields are, therefore, not necessarily isolated and, in fact, often are closely related to each other through combinatorial problems of the above type. For instance, Burstall[3.6] has formulated the cryptarithm problem as one of satisfying integer linear equalities with some added conditions of set inclusion and has developed a branch and bound technique for the machine solution of the problem. Following the success of this approach he has applied a similar technique to the problem of designing electricity networks. The heuristic methods developed in Part 3 and the above are direct applications of the studies in the field of artificial intelligence.

There are, however, problem areas where good algorithms may be obtained by some fundamental analysis, for instance the transmission problem reported in Part 1 of the thesis. Although this is suitable for a tree search(heuristic) approach and indeed some of the programs developed by other workers are along these lines, it may still be claimed that the dynamic programming algorithm developed ensures a better degree of optimality and is a superior strategy to adopt.

It is hard to decide whether tree search is an algorithm or a heuristic and an attempt is made to pursue this point in the next section. However, the generality and power of the tree search methods should be appreciated by the practitioners in the field. Falk and Soland[3,26] have recently outlined an algorithm of this type for general nonconvex programming, and Lawler and Wood [3.27] have surveyed the field quite extensively. Finally, to illustrate how the problems in these two areas are isomorphous to each other, consider the special problem of integer quadratic programming used for the optimal ordering of

actions as discussed by Dem'ianov and Khudiakov[3.28]. This problem is the same as the cryptarithm problem of Burstall, except for a quadratic criterion function. The algorithm developed in Chapter 7 can solve this problem with minor modifications.

c. What is a heuristic?

Heuristic is defined in the Shorter Oxford Dictionary as 'something serving to discover; heuristic method- a system of education under which the pupil is trained to find out things for himself'. Unfortunately, the term has been used in a less specific sense with consequent confusion as to its exact meaning. For instance, an accepted definition is:'a heuristic method is one that seeks to obtain good solutions for a small fraction of the cost which would be involved in obtaining optimal solutions'([1.23]). The heuristic methods developed by Cooper and Drebes[3.11] and Burstall[3.24] are indeed in this spirit. The formulation and solution of the simplified road design problem(Chapter6), may also be claimed to be in this sense a heuristic approach which makes use of mathematical programming to solve the subproblems generated.

A method can be claimed to be truly heuristic when sufficient consideration is given to the learning aspect of the computing scheme . The heuristic scheme of Reiter and Rice[3.23] in this respect does justice to the term, since use is made of the numerical information gained during the search to devise a learning scheme and hence a stopping rule.

There are, in general, two important questions related to the use of heuristic methods:

(i) what is one trying to achieve?

(ii) how well is one progressing?

To answer these an evaluation function is defined([1.23]) to measure the distance of the current state from the goal and some kind of 'performance index' is set up to measure the efficiency of the search strategies used. Evaluation function and performance index are highly context dependent terms. However, the

general principle remains that in a worthwhile heuristic scheme one should try to define these two quantities. For instance, in the branch and bound scheme for integer programs the objective function itself can be used as the evaluation function and its value may be taken as the basis of choosing the next problem (node) to investigate. Alternatively, one may define an evaluation function which consists not only of the objective function but also includes some term to measure the degree of integer infeasibility, that is the number of variables which are not currently integral.

As for performance, this is an item often neglected by many workers. To measure the efficiency of tree search one should make use of two pertinent variables, storage and computing time. The computing cost in a multiprogramming machine is a function of these two quantities.

An important concept that has emerged from the present study of heuristic methods is that, assuming the necessary computing facilities the heuristic techniques developed in Part 3 are finite. This is a very favourable point in support of this approach, contrasting with the works of Reiter and Rice[3.23] or Cooper and Drebes[3.11]. Within the framework of heuristic techniques finiteness of the search merits careful consideration. Analysis of such finiteness and the efficiency of the search techniques proposed are dependent primarily on the structure of the model being investigated; the importance of understanding and making use of such structures is discussed in the next section.

d. Solution techniques and their dependence on the structure of the mathematical model.

In developing solution techniques for various optimization models it is only logical to take advantage of the special properties of the model. A typical example is to be found in the solution of the transportation problem by the U-V method or the network flow algorithm, which do away with the multiplications and divisions involved in the full simplex method.

The production scheduling problem or the economic lot size problem, formulated as the linear fixed charge problem(Chapter 9) also serves to illustrate

this point. The technique developed in Chapter 9 takes advantage of some, but not all, of the properties of the model. This method is quite powerful in solving these problems, whereas the technique of Cooper and Drebes[3.11] fails to produce the optimal solution even for a small problem taken from Hadley[2.2]. There are two important properties of this model:

(i) The constraint matrix is unimodular, like the transportation problem.

(ii) The principle of optimality may be applied to deduce the optimal policy through successive periods of time.

Wagner and Whitin[3.29] have developed a very efficient dynamic programming algorithm making use of these two properties. The success of their algorithm stems from the fact that only the basic operations of addition, subtraction, and comparison are involved. In the absence of the structure mentioned above nothing can be said a-priori as to the relative efficiency of the various methods of solving ordinary fixed charge problems. The heuristic scheme developed in Chapter 9, may be claimed to be a very promising method.

Appendix 1. B.E.S. Specifications (1960) for 400kv L.6. Lines

The more important details of the B.E.S. specifications for 400kv L.6. lines are described in this appendix. The information and its interpretation was provided by CEGB [1.38].

1) The height to bottom conductor on a standard height tower of this design is 65' 3". Thus with 'M' representing minus height and 'E' representing plus height a D.M20' is 45' 3" to the bottom conductor whilst a D.E20' is 85' 3". The height to bottom conductor on all standard height angle towers is also 65' 3" and the extensions and reductions are represented in the same way by 'M' and 'E'. The maximum range of extensions for a D tower is minus 20 ft to plus 70 ft whilst those of angle and terminal towers are minus 20 ft to plus 40 ft.

2) Each conductor is tensioned such that the maximum working tension with ½" radial thickness of ice and a transverse wind load of 8 lbs, temperature 22°F does not exceed 12,000 lbs. This means that there is a total of 48,000 lbs per phase terminal loading. The maximum working tension in each conductor in still air at 60°F, must not exceed 5,850 lbs. A single earth conductor of the same characteristic is used.

3) The design span is 1,200 ft giving 25 ft statutory ground clearance at 122°F in still air on level ground with standard height towers. This represents a sag of 40' 3". With the 3 ft additional ground clearance now required to compensate for conductor 'creep' the minimum conductor clearance to ground is 28 ft. A design span of 1200 ft is therefore not obtainable with standard height towers.

4) The maximum sum of adjacent spans may total 2,640 ft provided that no single span shall normally exceed 1,800 ft. The maximum dead-weight of ice loaded conductors on any tower shall not exceed 2,400 ft

(1,800 ft for a terminal tower). On straight line towers ('D' type) the weight of conductors carried by a suspension insulator shall not be less than 35% of the sum of the adjacent spans at a temperature of 22°F without wind or ice loading.

5) The clearances from the various objects, structures, and the terrain, should conform to the values given in table A 1.1.

Feature	Amount of Clearance (ft)	Remarks
Ground	28'	Includes, tracks, footpaths, fences etc.,
Existing Railways	32'	
Railways (with electrification)	48'	Future electrification to be considered
Railway Property	48'	Where mobile cranes operate
Offending Structure	21'	Includes wall, scaffolding etc.
Metal Building	29'	
Roads	31'	Ordinary metalled highways
Trees	18'	Where noted there must be retained
Orchards	28'	Hops and vines where spraying equipment is used
Power lines	18'	Distribution lines

Table A 1.1

6) In grounds with side slopes and for structures offset to the line the following interpretation of the clearance should be adopted.

The line can swing upto a maximum angle of $\theta=45^{\circ}$ with the vertical direction. Let R_s denote the radius which is centred at hinge(s), representing the swing of the line due to a sidewind; these are shown in the elevation of the line Fig. A 1.1 and side elevation Fig. A 1.2. The configurations which must be considered for the computation of clearance under these conditions are shown in Fig.A 1.2, A 1.3, A 1.4. The hinge point, hinge(s) is obtained by linear interpolation between the points of support, i.e. P,Q and sag is calculated considering the extra loading due to wind.

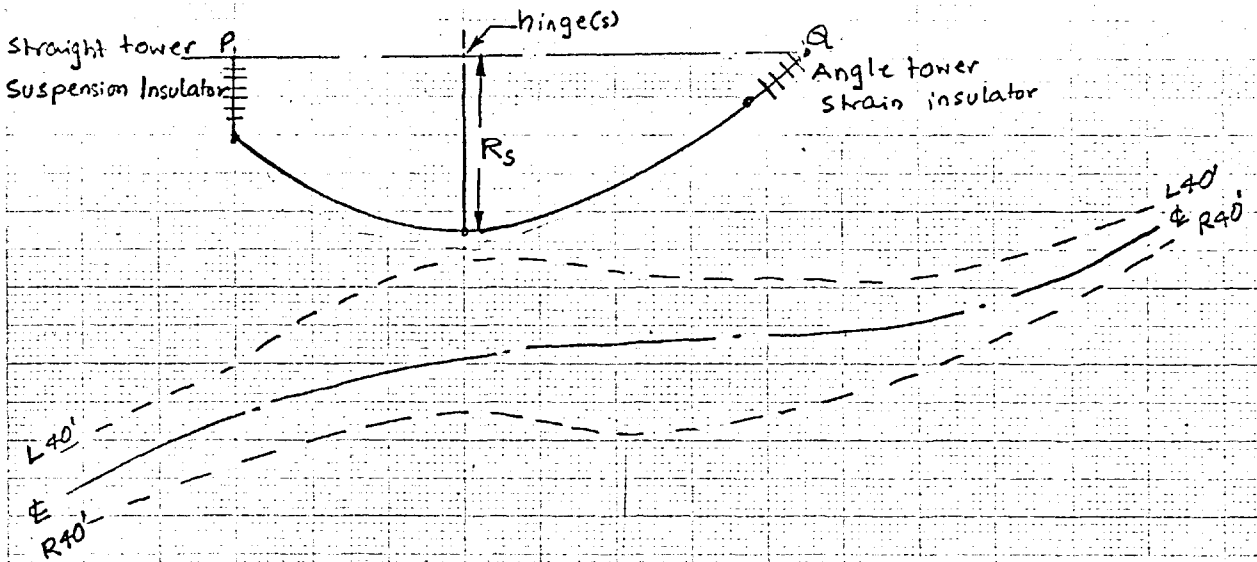
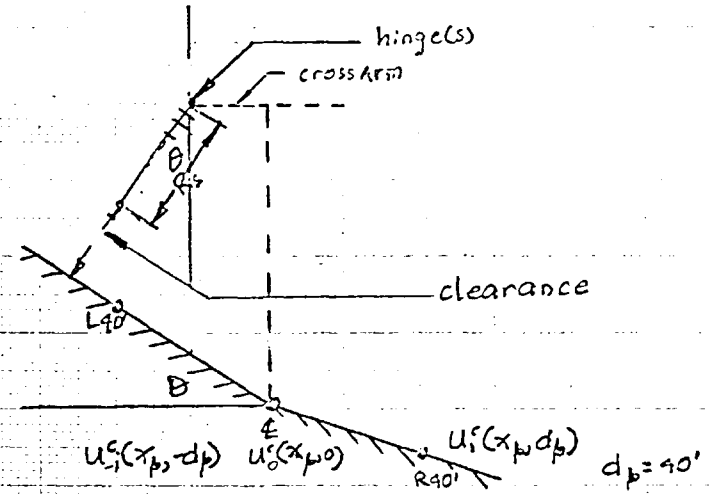
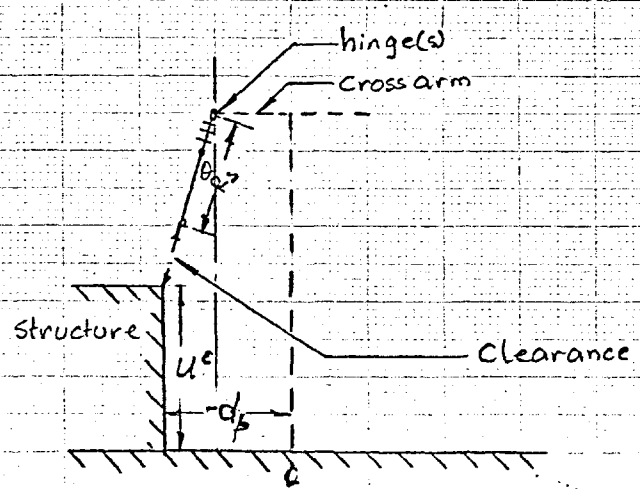


Fig. A.1.1



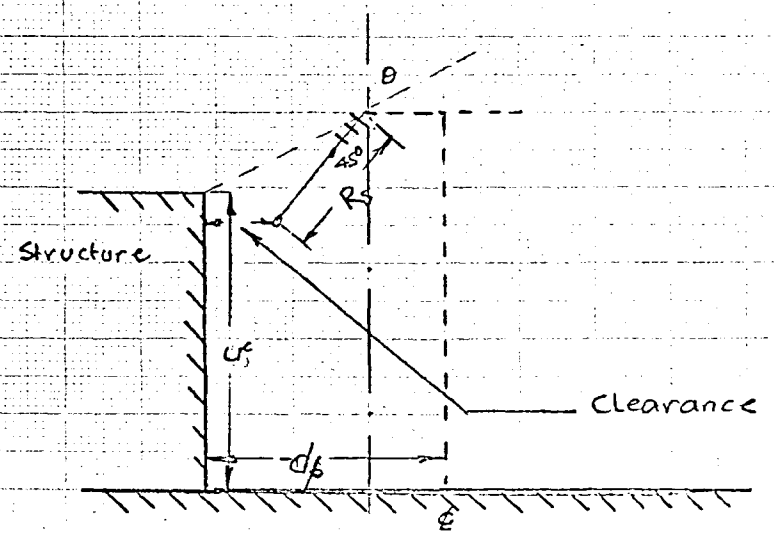
Clearance for side slopes

Fig. A.1.2.



$\theta < 45$ Clearance from structure offset to the line

Fig. A.1.3



$\theta > 45$ Clearance from structure offset to the line

Fig. A.1.4.

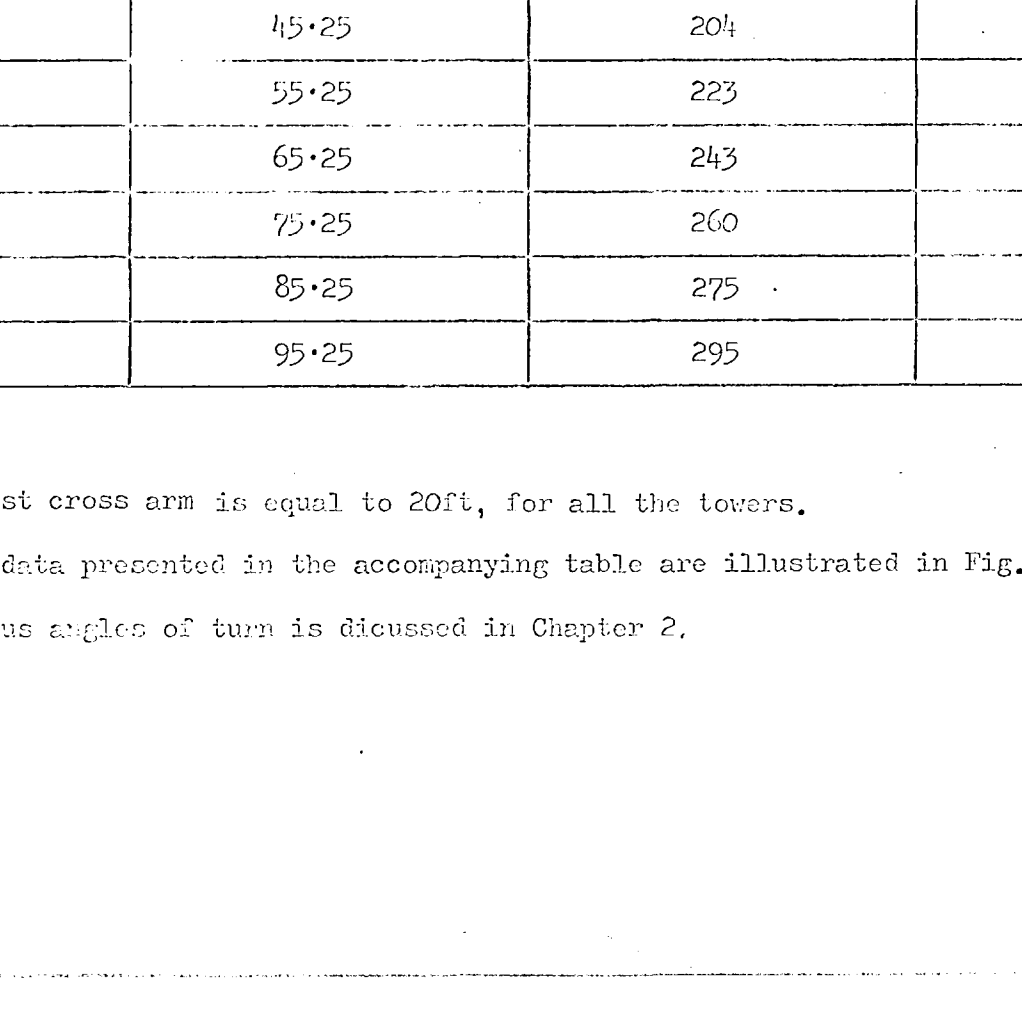
Appendix 2

Design Data for the L-6 Towers Used

TOWER TYPE	TOWER WIDTH AT BASE SETTING LEVEL W_k (Ft)	SUSPENSION HEIGHT FROM THE BASE SETTING LEVEL H_k (Ft)	COST IN PERCENT OF THE CHEAPEST TOWER C (H_k)	CODING NO.	REMARKS
D.M20'	29·875	45·25	100	1	Suspension towers insulator string length $18\frac{1}{2}$"
D.M10'	32·938	55·25	104	2	
D.STD	36·000	65·25	111	3	
D.E10'	39·063	75·25	116	4	
D.E20'	42·125	85·25	124	5	
D.E30'	45·188	95·25	129	6	
D.E40'	48·250	105·25	137	7	
D.E50'	51·312	115·25	143	8	
D.E60'	54·250	125·25	151	9	
D.E70'	57·250	135·25	157	10	
D30.M20	34·500	45·25	210	11	Angle towers maximum angle of turn = 30° Uses strain insulators
D30.M10	39·250	55·25	218	12	
D30.STD	44·000	65·25	232	13	
D30.E10	48·750	75·25	242	14	
D30.E20	53·500	85·25	258	15	
D30.E30	58·250	95·25	268	16	
D30.E40	63·000	105·25	284	17	
D60.M20	41·469	45·25	240	21	Angle towers maximum angle of turn = 60° Uses strain insulators
D60.M10	52·485	55·25	255	22	
D60.STD	47·500	65·25	271	23	
D60.E10	52·516	75·25	283	24	
D60.E20	57·537	83·25	305	25	
D60.E30	62·552	95·25	314	26	
D60.E40	67·500	105·25	324	27	
DTM20	34·500	45·25	204	31	Terminal Tower Uses strain insulators
DTM10	39·250	55·25	223	32	
DTSTD	44·000	65·25	243	33	
DTE10	48·750	75·25	260	34	
DTE20	53·500	85·25	275	35	
DTE30	58·250	95·25	295	36	

lc, the length of the lowest cross arm is equal to 20ft, for all the towers.

The symbols for the data presented in the accompanying table are illustrated in Fig. A 2.1, the choice of the angle towers for the various angles of turn is dicussed in Chapter 2.



Side view.

Fig. A.2.1.

Appendix 3.

Duality Theory of Linear Programming([1.4], [2.1], [2.3], [2.5]) and Primal (Dual) Algorithm.

Consider the two linear programming problems,

Problem A. Maximize $f(x) = p_1 x_1 + p_2 x_2 + \ldots p_n x_n$

subject to,

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &\le b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &\le b_2 \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &\le b_m \end{aligned} \qquad \text{(A 3.1)}$$

and $x = (x_1, x_2, \ldots x_n)' \ge 0$. In matrix notation this can be written as,

maximize $f(x) = p'x$,

subject to,

$$A x \le b \ , \ x \ge 0 \qquad \text{(A 3.2)}$$

and,

Problem B. Minimize $G(v) = v_1 b_1 + v_2 b_2 + \ldots + v_m b_m$

subject to,

$$\begin{aligned} a_{11}v_1 + a_{21}v_2 + \ldots + a_{m1}v_m &\ge p_1 \\ a_{12}v_1 + a_{22}v_2 + \ldots + a_{m2}v_m &\ge p_2 \\ a_{1n}v_1 + a_{2n}v_2 + \ldots + a_{mn}v_m &\ge p_n \end{aligned} \qquad \text{(A 3.3)}$$

and $v \ge 0$, or is matrix notation

minimize $G(v) = b'v$

subject to,

$$A'v \ge p \ , \ v \ge 0 \qquad \text{(A 3.4)}$$

These two problems are said to be dual to each other, and if the problem A is called primal then problem B is called dual. For this pair of problems duality theory states that,

1. the cost of any feasible solution to problem A must be less than or equal to the cost of any feasible solution to the problem B, that is $f(x) \leq G(v)$, (A 3.5)

2. if there exist feasible solutions to each of these problems, then there must be an optimum solution to each of these problems such that $f_{max}(x) = G_{min}(v)$

3. if $y_1 = x_{n+1}$, $y_2 = x_{n+2}$, ... $y_m = x_{n+m}$ denote the slack variables of the primal problem A, and u_1, u_2, ... u_n the surplus variables of the dual problem B, then the primal and the dual variables in the optimal solution must satisfy the condition of complementary slackness, that is

$$\sum_{i=1}^{m} y_i v_i + \sum_{j=1}^{n} x_j u_j = 0 \qquad \text{(A 3.6)}$$

and $y, v, x, u, \geq 0$. (A 3.7)

If one re-writes the v_i as u_{n+i}, i=1,2,...m, then (A 3.6) and (A 3.7) can be re-written as

$$\sum_{j=1}^{n+m} x_j v_j = 0 \qquad \text{(A 3.8)}$$

and $x_j, u_j \geq 0$, j=1,2,... n+m, (A 3.9)

4. if any of the primal variable x is unrestricted in sign then the corresponding dual variable is constrained to be zero, that is, the corresponding dual constraint is an equality and vice versa,

5. if the solution of the primal problem is infeasible that of the dual problem is unbounded and vice versa.

All the existing simplex algorithms of linear programming make use of the above properties in one form or another.

Representing the problem A in the Tucker-Beale tableau form, where

$p_j = a_{oj}$ and $b_i = a_{io}$, and introducing a set of trivial equiations in the form $x_j = -(x_j)$, $j=1,2,\ldots,n$, one obtains the following tableau. Let α_{ij}, $i = 0,1,2,\ldots n+m$, and $j=0,1,\ldots n$, represent any element of the tableau, then the primal (dual) simplex rules for obtaining the pivot elements for the pivotal transformation (Appendix 5) to be effected on this tableau, can be stated as follows

	$-x_1$	$-x_2$	$-x_n$
x_0	$a_{00} = \alpha_{00}$	$-a_{01} = \alpha_{01}$	$-a_{02} = \alpha_{02}$	$-a_{0n} = \alpha_{0n}$
x_1	$0 = \alpha_{10}$	$-1 = \alpha_{11}$	$0 = \alpha_{12}$	$0 = \alpha_{1n}$
x_2	0	0	-1	0
x_n	0	0	0	-1
x_{n+1}	$a_{10} = \alpha_{n+1,0}$	$a_{11} = \alpha_{n+1,1}$	$a_{12} = \alpha_{n+1,2}$	$a_{1n} = \alpha_{n+1,n}$
x_{n+2}	$a_{20} = \alpha_{n+2,0}$	$a_{21} = \alpha_{n+2,1}$	$a_{22} = \alpha_{n+2,2}$	$a_{2n} = \alpha_{n+2,n}$
x_{n+m}	$a_{m0} = \alpha_{n+m,0}$	$a_{m1} = \alpha_{n+m,1}$	$a_{m2} = \alpha_{n+m,2}$	$a_{mn} = \alpha_{n+m,n}$

If the problem is primal [dual] feasible, that is $\alpha_{i0} \geq 0$, $i \geq 1$ [$\alpha_{0j} \geq 0, j \geq 1$], choose a column q [rowp], with first element α_{0q}[α_{p0}] negative. From among the positive [negative] elements in this column [row] select the one for which the ratio α_{io}/α_{iq}[α_{oj}/α_{pj}] attains its least value. This is the pivot element α_{pq}. If no positive [negative] element exists in such a column [row] then the problem is unbounded [infeasible]. The optimal solution is obtained when both primal and dual feasibility is attained.

Appendix 4. Duality Theory of Quadratic Programming ([1.10]).

The original duality theorem due to Dorn [1.10] was stated in a form in which the primal program considered was a minimization problem. The theorem stated here is again due to him and is to be found in Saaty and Bram pp 123 [1.14]. Consider the problems,

$$\text{A.} \quad \text{Maximize } f(x) = \sum_{j=1}^{n} p_j x_j - \tfrac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_i x_j$$

subject to the constraints,

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \ , \quad i=1,2,3,\ldots m \qquad \text{(A 4.1)}$$

and $x = (x_1 x_2, \ldots x_n)' \geq 0$

where c_{ij} are the elements of a symmetric matrix whose quadratic form is positive (semi) definite. In matrix notation this can be written as

$$\text{Maximize } f(x) = p'x - \tfrac{1}{2} x'Cx$$

$$\text{subject to } Ax \leq b \text{ and } x \geq 0 \qquad \text{(A 4.2)}$$

$$\text{Problem B.} \quad \text{Minimize } G(u,v) = \sum_{j=1}^{m} b_j v_j + \tfrac{1}{2} \sum_{j=1}^{n} \sum_{j=1}^{n} c_{ij} u_i u_j$$

subject to

$$\sum_{i=1}^{m} a_{ij} v_j + \sum_{i=1}^{n} c_{ij} u_i \geq p_j \ , \quad j=1,2,\ldots n \qquad \text{(A 4.3)}$$

and $v = (v_1, v_2, \ldots v_n)' \geq 0$

In matrix notation this is written as

$$\text{Minimize } G(u,v) = \tfrac{1}{2} u'Cu + b'v$$

$$\text{subject to } A'v + Cu \geq p \qquad \text{(A 4.4)}$$

and $v \geq 0$

The problem A is known as the primal problem and problem B the dual problem. If p and A are taken with negative signs and one considers the primal minimization problem, then one obtains the Type I problem due to Dorn [1.10]. Further, setting C=0 one obtains the duality theorem of linear programming.

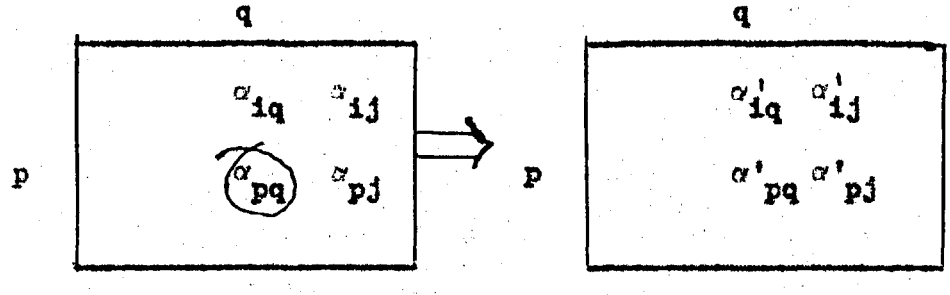
The properties (1) to (4) of the primal and the dual problems stated in

Appendix 3, also apply to the two problems considered here. However, if C is positive definite, then there will always be a solution to the dual problem, obtained by setting v= 0 and solving for Cu= p, thus the primal problem can never be unbounded. But the primal problem can be infeasible, and corresponding to this infeasibility, in the v direction one may decrease the linear part of the dual functional indefinitely and hence the solution will be unbounded.

To apply the primal algorithm one needs to start from a primal feasible solution. To apply the dual algorithm one needs to start from a dual feasible solution. The setting up of the tableau, a variation on the primal algorithm and the pivot choice rules are considered in Chapter 5.

Appendix 5. Rules for Pivotal Transformations ([2.23]) on the Linear and Quadratic Programming Tableaus.

Linear Programming Tableau.



(A 5.1)

Tableau A 5.1 Tableau A 5.2

When a pivot operation is performed on the element α_{pq} of the original tableau, the elements of the transformed tableau are given by the following rules, (for the Tucker-Beale representation)

$$\left.\begin{aligned} \alpha'_{pq} &= -1 && \text{for the enlarged tableau} \\ &= -1/\alpha_{pq} && \text{for the condensed tableau} \\ \alpha'_{pj} &= 0 && \text{for the enlarged tableau} \\ &= \frac{\alpha_{pj}}{\alpha_{pq}} && \text{for the condensed tableau} \end{aligned}\right\} \quad j=0,1,\ldots n \text{ and } j\neq q \qquad \text{(A 5.2)}$$

$$\alpha'_{iq} = -\frac{\alpha_{iq}}{\alpha_{pq}} \quad , \; i \neq p \text{ for the enlarged and the condensed tableau,}$$
$$i=0,1,\ldots m \quad \text{(condensed)}$$
$$i=0,1,\ldots n+m \quad \text{(enlarged)}$$

$$\alpha'_{ij} = \alpha_{ij} - \alpha_{iq} \times \alpha_{pj} / \alpha_{pq} \quad \text{for both enlarged and the condensed tableau.} \qquad \text{(A 5.2) continued.}$$

for all i,j excepting i=p or j=q.

Quadratic Programming Tableau.

On a quadratic programming tableau two types of pivot operations are performed, single pivot and double pivot.

Single pivot.

This is always performed on the diagonal element of the symmetric/skew symmetric tableau and this property of the tableau is preserved after the transformation. In the tableau A 5.3 if one pivots on the element α_{pp} then the elements of the transformed tableau can be trivially obtained using (A 5.2). However, to apply the pivot rules (A 5.2) it is necessary to know the pth row and the pth column of the tableau. The pth row and p^{th} column are best created explicitly by employing the symmetry/skew symmetry property described in Section b Chapter 5. Having obtained the row and the column (note these are skew symmetric), the transformation rules (A 3.2) for condensed tableaus can be applied explicitly. Such a transformation is indicated between tableaus A 5.3 and A 5.4, but not elucidated in any further detail. For efficiency in computer implementation, the actual procedure incorporates suitable row and column multipliers, the details of which can be found in the procedure 'pivot', of the code.

	x_0	$-x_1$	$-x_p$	$-u_{n+1}$	$-u_{n+n}$
u_0	α_{00}		α_{p0}		
u_1	α_{10}	α_{11}	α_{p1}		
u_p	α_{p0}	α_{p1}	α_{pp}	$-\alpha_{n+1,p}$	$-\alpha_{n+n,p}$
x_{n+1}	$\alpha_{n+1,0}$	$\alpha_{n+1,1}$	$\alpha_{n+1,2}$	$\alpha_{n+1,n+1}$	
x_{n+n}	$\alpha_{n+n,0}$	$\alpha_{n+n,1}$	$\alpha_{n+n,2}$	$\alpha_{n+n,n+1}$	$\alpha_{n+n,n+n}$

Tableau A 5.3

Apply the pivot rules given in (A 5.2), having created the pth row and column

	x_0	$-x_1$	$-u_p$	$-u_{n+1}$	$-u_{n+n}$
u_0	α'_{00}				
u_1	α'_{10}	α'_{11}			
x_p	α'_{p0}	α'_{p1}	α'_{pp}		
x_{n+1}	$\alpha'_{n+1,0}$	$\alpha'_{n+1,1}$	$\alpha'_{n+1,2}$	$\alpha'_{n+1,n+1}$	
x_{n+n}	$\alpha'_{n+n,0}$	$\alpha'_{n+n,1}$	$\alpha'_{n+n,2}$	$\alpha'_{n+n,n+1}$	$\alpha'_{n+n,n+n}$

Tableau A 5.4

Double pivot.

This is equivalent to block pivoting in a 2 × 2 matrix, but the transformation equations in matrix notations are not of great convenience from the view-point of computation. Explicit transformation equations for each element are given in [2.15] and will not be repeated here. An alternative method of creating only two relevant rows and columns of an intermediate tableau, is described, which it may be claimed, is a logical approach from the consideration of ease of coding and perhaps justifies the marginally extra storage space used. Let in the tableau A 5.5, u_p

and x_q be required to exit the basis, x_p and u_q to enter it. The effect of double pivot can be achieved if one first pivoted on the element α_{qp} and then pivoted on the element α'_{pq} of the subsequent tableau. The elements α''_{ij} of the transformed final tableau after these two pivot steps will be symmetric/skew symmetric, however, the intermediate tableau if one explicitly obtained it will not possess this property. Let two rows and columns rp, rq, cp, cq, be created, Fig. A 5.6 employing the symmetry property outlined in Section b Chapter 5, to represent the pth and q^{th} rows and columns of the tableau A 5.5. Let two further rows and columns srp, srq, scp, scq, be created corresponding to those of an intermediate tableau which one would obtain if one pivoted in the element α_{qp}. Assuming such a pivot operation has been carried out on α_{qp}, employing the rules (A 5.2) and using the elements in Fig. A 5.6, the elements of such rows and columns (Fig. A 5.7) will be given by the equations.

$$\alpha'_{scp.i} = -\alpha_{ip}/\alpha_{pq} \qquad i \neq q$$

$$\alpha'_{scp.q} = 1/\alpha_{qp}$$

$$\alpha'_{srp.j} = \alpha_{pj} - \alpha_{pp}.\alpha_{qj}/\alpha_{qp} \qquad j \neq p$$

$$\alpha'_{srp.p} = \alpha'_{scp.p} = \alpha_{pp}/\alpha_{pq}$$

$$\alpha'_{scq.i} = \alpha_{iq} - \alpha_{ip}.\alpha_{qq}/\alpha_{qp} \qquad i \neq q$$

$$\alpha'_{scq.p} = \alpha_{qq}/\alpha_{qp}$$

$$\alpha'_{srq.j} = \alpha_{qj}/\alpha_{qp} \qquad j \neq p$$

$$\alpha'_{srq.p} = 1/\alpha_{qp}$$

{ The right hand side elements of these equations where necessary should be interpreted as equivalent to the symmetric and skew symmetric elements. }

(A 5.3)

	x_o	$-x_1$	$-x_p$	$-u_q$	$-u_{n+m}$
u_o	α_{oo}				
u_1	α_{1o}	α_{11}			
u_p	α_{po}	α_{p1}	α_{pp}		
x_q	α_{qo}	α_{q1}	α_{qp}	α_{qp}	
x_{n+m}	$\alpha_{n+m,o}$	$\alpha_{n+m,1}$	$\alpha_{n+m,p}$	$\alpha_{n+m,q}$	$\alpha_{n+m,n+m}$

Tableau A 5.5

	c_p	c_q	
rp	α_{po} α_{p1}	α_{pp}	$-\alpha_{qo}$ $-\alpha_{q1}$ $-\alpha_{qp}$ $-\alpha_{n+m,p}$
rq	α_{qo} α_{q1}	α_{qp}	α_{qq} $\alpha_{n+m,q}$
		$\alpha_{n+m,p}$	$\alpha_{n+m,q}$

Fig. A 5.6

	scp	scq	
srp	$\alpha'_{srp,o}$ $\alpha'_{srp,1}$	$\alpha'_{scp,p}$	$\alpha'_{scq,o}$ $\alpha'_{scq,1}$ $\alpha'_{scq,p}$ $\alpha'_{srp,n+m}$
srq	$\alpha'_{srq,o}$ $\alpha'_{srq,1}$	$\alpha'_{scp,q}$	$\alpha'_{scq,q}$ $\alpha'_{srq,n+m}$
		$\alpha'_{scp,n+m}$	$\alpha'_{scq,n+m}$

Fig. A 5.7

Imagine that one now pivots on the element $\alpha'_{scq.p}$ or $\alpha'_{srp.q}$ which are in fact the representations of the same element α'_{pq} of the intermediate tableau. The elements of the final tableau A 5.8 are then given by the equations,

$$\alpha^{11}_{ij} = \alpha_{ij} - \alpha_{ip}\alpha_{qp}/\alpha_{qp} - \alpha'_{srp.j}\,\alpha'_{scq.i}/\alpha'_{scq.p}$$

$i \neq p$ or q

$$\alpha^{11}_{qj} = \alpha'_{srp.j}/\alpha'_{scq.p} \qquad \text{(A 5.4)}$$

$j \neq q$

$$\alpha^{11}_{qq} = 1/\alpha'_{scq.p}$$

$$\alpha^{11}_{pj} = \alpha'_{srq.j} - \alpha'_{srp.j}\,\alpha'_{scq.q}/\alpha'_{scq.p}$$

$j \neq q$

$$\alpha^{11}_{pq} = -\alpha'_{srq.q}/\alpha'_{scq.p}$$

In the equations (A 5.4) and in the final tableau, the rows p and q are interchanged so that the column and row ordering of the variables are preserved. The actual coding of this procedure called 'duopivot' can be found in Appendix 9 where, for convenience of interpretation it is noted, p and q are replaced by the identifiers hi and lo.

	x_o	$-x_1$	$-u_p$	$-x_q$	$-u_{n+m}$
u_o	α^{11}_{oo}				
u_1	α^{11}_{1o}	α^{11}_{11}			
u_p	α^{11}_{po}	α^{11}_{p1}	α^{11}_{pp}		
u_q	α^{11}_{qo}	α^{11}_{q1}	α^{11}_{qp}	α^{11}_{qq}	
x_{n+m}	$\alpha^{11}_{n+m,o}$	$\alpha^{11}_{n+m,1}$	$\alpha^{11}_{n+m,p}$	$\alpha^{11}_{n+m,q}$	$\alpha^{11}_{n+m,n+m}$

Tableau A 5.8

APPENDIX 6

```
JOB
SC07001201,CEGB TOWER SPOTTING (DATA PROCESSING) MITRA
INPUT
1 LINE ROUTE DATA
2 EXTRA VARIABLES
OUTPUT
0 LINEPRINTER 155\\0 LINES
TAPE
1 L0083
STORE 32 BLOCKS
COMPUTING 8000\\ INSTRUCTIONS
EXECUTION 8.0 MINUTES
COMPILER EMA
MAIN -> 4000
AUXILIARY(0,0)
DEPTH0
DUMPS0


IGNORE TRACE
```

```
ROUTINE 100
>> THIS IS A STRAIGHT READ ROUTINE INPUTS 5 WORDS AND 3WORDS/CARD AND
   READS A CARD UPTO NEWLINE


J= 1(1)5
READ(;J)
JUMP 2,M'≠ØCODE(()
JUMP 2,J<3
;5=;3
;4=;2
;3=;1
;2=;1
JUMP 1
2) REPEAT
1) READCH(Q')
JUMP 1,Q'≠ØCODE(NL)
RETURN
**
ROUTINE101
| the interpolation routine now followws

START TRACE
PRINTLINE
QUERY PRINTS FROM INTERPOLATION RCU\\TINE FOLLOWS
S=1
R=0
B(2S)=B(2S) + C'
PRINTLINE
OUTPUT OF STARTING LOCATION
PRINT(B(2S))2,5
20)S=S + 1
JUMP80,B(2S)> -1
STOP TRACE
RETURN

80)PRINT(S)3,0
T=\\0
T'=0
21)R=R + 1


JUMP21,A(5R)<B(2S)
JUMP40,A(5R - 1)>100 | special clearance is being checked
JUMP22,A(5R-6)=\\18\\     | line crossings excluded
JUMP22,A(5R-5)<B(2S) - 150 | the point too distant for interpolation
JUMP22,A(5R-6)> 100     | last site was a special clearance site
JUMP23
```

```
22)T'=1
23)JUMP60,B(2S+2)<-1
JUMP25,A(5R + 4)= 18 | the next site is a power line crossing
JUMP25,A(5R+5)>B(2S) + 150
JUMP25,A(5R + 4)> 100  | NEX SITE IS SPECIAL CLEARANCE
JUMP50

25)T=1


50)JUMP29,T'≠1
JUMP40,T=1
JUMPDOWN26    | backward extrapolation
31)JUMPDOWN(R102)
JUMP20


29)JUMP30,T≠1
JUMPDOWN27        | forward extrapolation
JUMP31


30)T=0(1)2
T'=T-1


;(2T'+3)=A(5R+5T'-4)
;(2T'+4)=A(5R + 5T' -2)
;(10+T'+2)=A(5R+5T')
REPEAT


JUMP31
26)T'=0(1)1
;(2T'+3)=A(5R+5T'-4)
;(2T'+4)=A(5R+5T'-2)
;(10+T'+2)=A(5R+5T')
REPEAT
;11=2;12-;13
;\\1=2;3-;5
;2=2;4-;6
RETURN


27)T=0(1)1
T'=T-1
;(2T'+3)=A(5R+5T'-4)
;(2T'+4)=A(5R+5T'-2)
;(10+T'+2)=A(5R+5T')
REPEAT
```

```
;13=2;12-;11
;5=2;3-;1
;6=2;4-;2
RETURN


40)PRINTLINE
INSUFFICIENT DATA FOR INTERPOLATION AT SITE=
PRINT(B(2S))4,0
JUMP20
60)JUMP61,D0>90          | no intrpln reqd for special trs or terminal trs
D1=45+ED0/2
D2=45-ED0/2
PRINTLINE
THE TWO ANGLES ARE (DEGREES)
PRINT(D1)2,1
SPACE4
PRINT(D2)2,1
PRINT('RESPECTIVELY')
D1=;D1/180       | lower angle in radian
D2=;D2/180       | upper  ''        ''
C1=C3 -(C3-C1)*(40ØTAN(D1))*.02| the elevation at the end
PRINT('C1')
PRINT(C1)4,2
C2=C4-(C4-C2)*(40ØTAN(D2))*.02  |  the elevation at the rhs end
PRINT('C2')
PRINT(C2)4,2
C7=C5-(C5-C7)*(40ØTAN(D1))*.02   | the elevation at the lhs end beyond
PRINT('C7')
PRINT(C7)4,2


C8=C6-(C6-C8)*(40ØTAN(D2))*.02 \\\| the elevation at the rhs end beyond
;1= B(2S-1) \\-(B(2S-1)-C1)*(56.5)*(ØCO\\S(D1))*(.025)
;2=B(2S-1) -(B(2S-1) -C2)*(56.5)*(ØCOS(D2))*(.025) |the pi set is for 40 ft grad
;3=B(\\2S-1) -(B(2S-1)-C7)*(56.5)*(ØCOS(D1))*(.025)
;4=B(2S-1) -(B(2S-1) -C8)*(56.5)*(ØCOS(D2))*(.025)


JUMPDOWN(R102/10)
C'=ØFRPT(B(2S))
PRINTLINE
THE INTERPOLATION GRADIENT AT THE START OF THE NEXT SECTION=
PRINT(C')2,5
JUMP20



61)PRINTLINE
THIS SECTION END\\S IN TERMINAL TOWER OR CONTINUOUS TOWER
;1=C3-(C3-C1)*0.8
;2=C4-(C4-C2)*0.8
;3=C5-(C5-C7)*0.8
;4=C6-(C6-C8)*0.8
```

```
JUMPDOWN(R102/10)
C'=ØFRPT(B(23))
JUMP20
**


ROUTINE 106
P=1
PRINTLINE
CLEARANCE DETAILS
NEWLINE
PRINTLINE
CHAINAGE          L40'          C/L          R40'          AMOUNT
NEWLINE
1)JUMP 3,A(P+3)≥100
SPACE1
PRINT(A(P+4))5,0
SPACE8
PRINT(AP)3,1
SPACE8
PRINT(A(P+1))3,1
SPACE7
PRINT(A(P+2))3,1
SPACE9
PRINT(A(P+3))2,1
NEWLINE
JUMP11
\\13)P=P+5
JUMP 1,A(P\\\\)≠-1
RETURN
11)C21=ØMOD(A(P+1)-AP)
C22=ØMOD(A(P+1)-A(P+2))
JUMP 12,(C(ØMAX(C0,21,22))/40)≤0.5
NEWLINE2
PRINTLINE
UNUSUAL SLOPE
NEWLINE
PRINT('CHAINAGE = ')
SPACE2
PRINT(A(P+4))5,0
NEWLINE


12)N=1(1)5
E'=ØTABLE(14,N)
E=E'-;4
JUMP 2,E<0.1
REPEAT
PRINTLINE
CLEARANCE NOT RECOGNISED
NEWLINE
2)JUMP 13
3)PRINTLINE
SPECIAL CLEARANCE , CHAINAGE=
PRINT(A(P+4))5,0
SPACE5
```

```
PRINT('OFF-SET = ')
SPACE10
PRINT('ELEVATION = ')
NEWLINE
JUMP 13
RETURN
14)INTEGER$28,21,18,3,32
**
\\\\\\\\\\\\\\\\

ROUTINE 105
R=1
PRINTLINE
TEST-SITE DETAILS
NEWLINE
CAPTION
CHAINAGE        TEST-SITE ELEVATION        GRADIENT/40ft        DIRECTION
NEWLINE
2)SPACE3
PRINT(B(R+1))5,0
SPACE12
PRINT(BR)4,0
SPACE16
C=ØFRPT(B(R+1))
S=100\\OC
JUMP 1,S≥100
PRINT(S)2,1
SPACE13
PRINT('+')
NEWLINE
JUMP 3
1)S=S-100
PRINT(S)2,1
SPACE13
PRINT('-')
NEWLINE
3)R=R+2
JUMP 2,B(R+1)≠-2
RETURN
**



ROUTINE102
| tower base interpolation
;1=;3+(40/(;12-;11))*(;1-;3)?
;2=;4+(40/(;12-;11))*(;2-;4)?
;3=;3+(40/(;13-;12))*(;5-;3)?
;4=;4+(40/(;13-;12))*(;6-;4)?
```

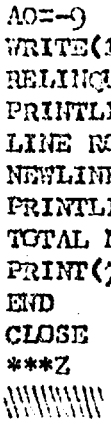
```
10)Y=;(MIN(;0,1,4))
NEWLINE
CAPTION
THE MINIMUM=
PRINT(Y)4,2
JUMP1,Y>B(2S-1)

2)Y=(B(2S-1)-Y)*0.001
B(2S)=B(2S)+Y
RETURN

1)JUMPDOWN2
Y=-2Y+0.1
B(2S)=B(2S)+Y
RETURN
**


TITLE
C.E.G.B. LINE ROUTE DATA PROCESSING PROGRAMME WITH MODS BY TFP
CHAPTERO
;->16
A->2561
B->1025
C->25
D->5
E->3

STOP TRACE


SELECT INPUT2
READ(D4)
READ(C')
READ(Q)
SELECT INPUT1




WIND TAPE(1,Q)



PRINTLINE
MAG-TAPE WRITING FROM BLOCK NO
PRINT(Q)3,0
Q=0
\100) K=-4
M=0

J'=1
```

```
1) READCH(I)
JUMP2,I=ØCODE(*)
\\JUMP2,I=ØCODE(-)
JUMP3,I=ØCODE(+)
JUMP5,I=ØCODE(>)
JUMP 99,I=ØCODE(()
JUMP 98,I=ØCODE())
JUMP9,I=ØCODE(])


PRINTLINE
WRONG CHARACTER PUNCH ON COLUMN 1 AT A CARD OF ABSCISSA >
PRINT(;5)
JUMPO


2) JUMPDOWN(R100)
K=K+5
J=1(1)5
JUMP11,J>3
JUMP11,;J<1500  | checks for the upper limit of the clearance elevation

PRINTLINE
ERROR IN CLEARANCE PROFILE AT X =
PRINT(;5)5,0
11)JUMP120,J≠5
;5=;5-D4
120)L=K+J-1
AL=;J
REPEAT




JUMP10,L<6


JUMP10,AL>A(L-5)          | this checks the ascending order of the clearance data
PRINTLINE
CARD OUT OF ORDER      X LOCATION =
PRINT(;5)5,0
10)JUMP1,I=ØCODE(-)


4) M= M + 1
JUMP20,;2\\<1500          | checks the upper limit of the test site elevation
PRINTLINE
ERROR IN TEST SITE PROFILE ( TOO LARGE ) AT X =
PRINT(;5)5,0
20)EM=;2
M=\\M+1
EM=;5?
```

```
JUMP12,M<3
JUMP12,EM>B(M-2)
PRINTLINE          | prints the error message if the test sites are not in
                                                        ascending sequence
ERROR IN TEST SITE PROFILE AT  X=
PRINT(EM)5,0
12)JUMP1



3)JUMPDOWN(R100)
C(J')=;1
C(J'+1)=;3
E=;2
J'=J' + 2
JUMP1

5)JUMPDOWN(R100)
PRINTLINE
DETAILS OF THE LINE
NEWLINE
PRINTLINE
SECTION NUMBER =
K'=1(1)2
DK'=;(K'+3)
70)DK'=DK'-100
JUMP 70,DK\\'>100
REPEAT
PRINT(D2)2,1
NEWLINE
PRINTLINE
SPAN LIMIT =
PRINT(;2)2,0
NEWLINE
PRINTLINE
STARTING ANGLE
SPACE12
PRINTLINE
END ANGLE =
PRINT(D1)2,1
SPACE2
PRINT('DEGREES')
NEWLINE
PRINTLINE
MAGNETIC TAPE BLOCKS USED =
PRINT(70+1)4,0
SPACE10
PRINTLINE
SECTION LENGTH =
PRINT(;1)2\\\5\\\,0
SPACE2
PRINT('TO')
SPACE2
PRINT(;3)5,0
D0=D1
NEWLINE
JUMPDOWN 102
JUMP 100
```

```
102)J=K+5          | sets up the index for end record marker
AJ=-1              | -1 is the end record marker for clearance records
L=J+1
K=J+5
J=L(1)K            | transfers the line details records to the end of clearance
                                                                    records
AJ=;(J-L+1)
REPEAT

J=M + 2
BJ= -2             | the test site end record marker


JUMPDOWN(R101)     | the inter pln routine now operates on the test sites
JUMPDOWN(R105)     |THIS WRITES OUT THE TEST-SITE DETAILS
J=1(512)513
WRITE(1,BJ)        | writes up the test site records
REPEAT


J=1(512)2049
WRITE(1,AJ)        | writes up the clearance and line details record
REPEAT


JUMPDOWN(R106)       |THIS WRIES OUT THE CLEARANCE DETAILS



Q=Q + 1

RETURN



99)M'=I
97)READCH(P')
JUMP 97,P'≠ØCODE(NL)
JUMP 1
98)M'=0
JUMP 97


0)READCH(P)
JUMP0,P≠ØCODE(NL)
T=T+1
PRINTLINE
FAULTY DATA AT
PRINT(;5)5,0
JUMP 1
9)J=0(1)512
AJ=0                    |to clear the main store location before writing up
                                                                  end marker

repeat
```

```
AO=-9
WRITE(1,AO)
RELINQUISH DECK(1)
PRINTLINE
LINE ROUTE DATA PROCESSING COMPLETED
NEWLINE
PRINTLINE
TOTAL NO OF BLOCKS OCCUPIED IN THE MAG TAPE =
PRINT(70 + 1)4,0
END
CLOSE
***Z
```



APPENDIX 7

```
JOB
LRR91PA1,MITRA C.E.G.B. TOWER-SPOT TEST 21/4/66
OUTPUT
0 LINEPRINTER 2500 LINES
INPUT
1 DATA FOR ROUTINES 31 AND 34
2 EXTRA CONSTANTS
TAPE
1 L0083
STORE 60/91 BLOCKS
COMPUTING 100000 INSTRUCTIONS
EXECUTION 25.0 MINUTES
COMPILER EMA


MAIN 32000
AUXILIARY(0,0)
DEPTH0
DUMPS0
IGNORE QUERIES


ROUTINE 1
>> DISTANCE TO THE MINIMUM AT COLD

        B=XL-XL'
        ;3=0.5[Y61-Y62]+0.5A'[BB]

        ;3=;3/(A'B)              | distance to the minimum

        RETURN
        **
ROUTINE2
| CLEARANCE PARAMETERS AT HOT TENSION ARE COMPUTED
| ALTERNATIVE ENTRY AT LABEL 30 FOR PRINT

JUMP31

30) U1= 500
Q=-1
1)Q=Q + 1
JUMP1,(U1+500)<B
U1=50(Q-1) + U1
U6= ØTABLE(120,Q-1)      | looks up table for hot tension
U7=ØTABLE(120,Q)         | for the next higher step
U6= U6 + 0.02(B-U1)*(U7-U6)      | interpolated tension
U0= 0.5(;10/U6)          | parameter of the hot sagging curve
;3= 0.5(Y61-Y62) + 0.5U0(BB)
;3\=;3/(U0B)     | distance to the minimum
```

```
RETURN
31)JUMPDOWN30

| The following block is for clearance computation

        JUMP 3
2)      R'=R'+1
        JUMP 3,A(5R'+4)<XL

        Q=0
        RETURN

|Look up accompanying sketches for clearance computation logic
| note use Q in actual run for run time officiency of the program

3)      U6=Y63+((A(5R'+4)-XL')/(XL-XL'))*(Y64-Y63)  | the hinge point

        W=Y61-U0(A(5R'+4)-XL')*(2;3-A(5R'+4)+XL')

        U7=U6-W                     | the swing radius
        JUMP 4,A(5R')<-1            | checks for special clearance
        JUMP 5,A(5R'+1)>(;/4)       | checks for excessive side slope

        JUMP 2,(U6-A(5R'))*ØCOS(A(5R'+1)) -8*W3*ØSIN(A(5R'+1))>U7+A(5R'+3)
| this checks for normal
                                                                | swing clearance

8)      Q=1

        RETURN
>> This block is for special clearance
4)      JUMP 8,U6≤A(5R'+2)          | checks if clearance point is above hinge point
        JUMP 6,(U6-A(5R'+2))<(A(5R'+1)-20-8W3)  | checks for swing greater
                                                  than 4545 degree

        U8=(U6-A(5R'+2))*(U6-A(5R'+2))+(A(5R'+1)-20-8W3)(A(5R'+1)-20-8W3) | sqrt of
                                                                            diagonal
        U8=ØSQRT (U8)
        JUMP 2,(U7+A(5R'+3))≤U8
        JUMP 8

6)      JUMP 2,(U6-A(5R'+2)+A(5R'+3))≤A(5R'+1)-20-8W3   | this checks for clearance
                                                  |in horizontal direction only


        JUMP 8

5)      U9=(U6-A(5R'))ØCOS(A(5R'+1))-8W3*ØSIN(A(5R'+1))  | this checks for clearance
                                                           for excessive side slope
```

```
        JUMP 2,U7*ØCOS(A(5R'+1)-;/4)+A(5R'+3)≤U9

        JUMP 8

120)    4378
        4375
        4464
        4542
        4610
        4670
        4723
        4769
        4810
        4847
        4880
        4909
        4935
        4959
        4980
        5000
        5017
        5033
        5048
        5061
        5073
        5084
        5095
        5104
        5113
        5122
        5128
        5135

90)B=XL-XL'
JUMPDOWN30
PRINTLINE
THE SOLUTION POINT OF CURRENT SPAN-Y61=
PRINT(Y61)4,1
CAPTION
Y62=
PRINT(Y62)4,1
CAPTION
Y63=
PRINT(Y63)4,1
CAPTION
Y64=
PRINT(Y64)4,1
PRINTLINE
HORIZONTAL TENSION=
PRINT(U6)4,0
PRINTLINE
TOWER TYPE /
PRINT(T)2,0
CAPTION
```

```
THE ABSOLUTE TOWER INDEX=
PRINT(T')3,0
PRINTLINE
AT DISTANCE    HINGE PT    SAG PT    ELEV'N    RADIUS    COS(THETA)
JUMP33


32)R=R-1

JUMP33,A(5R)>-1
PRINTLINE
THIS IS SPECIAL CLEARANCE LOCATION OFFSET=
PRINT(A(5R+1)) 3,2
CAPTION
ELEVATION=
PRINT(A(5R+2))3,2

33)NEWLINE
U6=Y63+((A(5R+4)-XL')/(XL-XL'))*(Y64-Y63)        | the hinge pt

W=Y61-U0(A(5R+4)-XL')*(2;3-A(5R+4)+XL')          | the sag pt

U7=U6-W          \\| the swing radius
SPACE3

PRINT(A(5R+4))4,0
SPACE 3
PRINT(U6)3,2
SPACE3
PRINT(W)3,2
SPACE3
PRINT(A(5R))3,2
SPACE3
PRINT(U7)3,2
SPACE3
PRINT(ØCOS(A(5R+1)))1,4
JUMP32,A(5R+4)>XL'
RETURN
**
```

```
ROUTINE 4
>> UNPACK INSTRUCTIONS FOR EACH STEP

        UNPACK (DM',0,K')
        P'=Z(100+M')
JUMP1,M'≠J
UNPACK (EJ',0,O')
        UNPACK (EJ,1,R')
        RETURN
        1)UNPACK(EJ,2,O')
UNPACK(EJ,3,R')

        RETURN
        **

ROUTINE 5
>> PACK INSTRUCTIONS FOR EACH STEP OF STORAGE

        PACK (DM',0,K')
        PACK (DM',1,S')
        Z(100+M')=P'
        PACK (DM',3,J)
JUMP1,M'≠N
        PACK (EN,0,L')
        PACK (EN,1,R)
        RETURN
1)PACK(EN,2,L')
PACK(EN,3,R)

        RETURN


        **



ROUTINE 6
| stop by stop output for test run

JUMP10,L<2


RETURN
10)NEWLINE
CAPTION
A'=
PRINT(A')2,8
JUMP2,J≤S
JUMP3,J>(S+1)
```

```
CAPTION
SUBOPTIMAL SOLUTIONS /// THE CHECK PARAMETERS
4)PRINTLINE
T-SITE  T-HEIGHT  INDEX  COST/CENT  PRSITE  INDEX  TOWERINDEX DIST/LEFT
NEWLINE
5)PRINT(XL)4,0
PRINT(ILJ)3,2
SPACE 2
PRINT(J)3,0
SPACE
PRINT((K'+X'))5,0
SPACE 2
PRINT(XL')4,0
SPACE
PRINT(L')3,0
PRINT(S')3,0
SPACE 3
PRINT(P')5,0
RETURN


3)JUMPDOWN5
RETURN

2)JUMP1,T≠0
CAPTION
OPTIMAL SOLUTION STEP
JUMPDOWN4
RETURN
1)JUMPDOWN5
RETURN

**
```

```
ROUTINE 7
>> RESTORE ROUTINE FOR FINAL STORAGE

        J'=3T
        K'=ØINTPT (ZT+0.0001)
        S'=ØINTPT (YJ'+0.0001)
        J'=J'+1
        P'=ØINTPT (YJ'+0.0001)
        J'=J'+1
        L'=ØINTPT (YJ'+0.0001)

        RETURN
        **


ROUTINE8
| UNPACK ROUTINE FOR OUTPUT PRINT
UNPACK(DM',0,K')
UNPACK(DM',1,S')
UNPACK(DM',3,T)
JUMP1,M'≠J
UNPACK(EJ,0,L')
UNPACK(EJ,1,R)
RETURN
1)UNPACK(EJ,2,L')
UNPACK(EJ,3,R)
RETURN

**
ROUTINE 9
|final solution storage routine


V'=N
NEWLINE
T=0(1)S
M'=L(S+1)+T
N=M'
J=T
JUMP1,W13<0.1
M'=M'+5000
J=S+T+1
1)JUMPDOWN(R7)
JUMPDOWN(R6)
JUMP3,(L-V')<-0.1
P'=4000
3)JUMPDOWN(R5)



2)REPEAT

N=ØINTPT(V'+0.001)
RETURN


**
```

```
ROUTINE 10
| sort routine to prune the solution


JUMP30,T>6
J=T
JUMP28
30)J=T-6-1
28)JUMP29,Q'=1
ZJ=0
RETURN

29)Q'=J
JUMP35,ZJ<1
33)Q'=Q'-1
JUMP34,ZJ>ZQ'
ZQ'=0
34)JUMP33,Q'>0
35)RETURN

**


ROUTINE 11
| routine to set up the suboptimal parameters

JUMP1,1760<XL
10)JUMP20,L=N
PRINTLINE
NO SOLUTION FEASIBLE AT
PRINT(XL)4,0
L=L+1
RETURN


20)END
1)W'=XL-1760
V'=L
2)L=L-1
JUMP2,XL≥W'
L=L+1
PRINTLINE
SUBOPTIMAL COMPUTATION TAKEN BACK TO
PRINT(XL)5,0

JUMP3,D'=L
D'=L
W'=1
W11=1
RETURN
3)PRINTLINE
SOLUTION FAILS TWICE AT THIS POINT
L=ØENTPT(V' +.01)
JUMP10
**
```

```
ROUTINE 32
>> BLOCK TO GENERATE STARTING VALUE


        P'=3000
        R=1

        T=O(1)S
        READ (K')
        H'=T
        V'=H
        H=H'
        JUMPDOWN (R5)
        H=POINTPT(V' +0.1)
        REPEAT
        READ(B6)
        READ(B4)

        RETURN
        **

ROUTINE 31
>> INPUT ROUTINE

        READ (B1)                 | minimum span
        READ (B2)                 | maximum span
        READ (B3)                 | double span
        READ (Y65)                | string length of L6 type towers
        READ(Y66)                 | string length of L 2 type towers

        READ (;10)                | weight per unit length at 122'F and no wind
        READ (;11)                | the design tension at 22'F + WIND + ICE
        READ (;14)                | the equivalent loading at 22'F + WIND + ICE

        A'=;14/(2;11)             | the coefficient of the parabola
        PRINTLINE
        PARAMETER OF THE COLD CURVE=
PRINT(A')2,8

        READ (I)                  | the angle tower types
        READ (S)                  | number of toers in each set of angle towers

        L=O(1)I

        L'=O(1)S

        Q'=L(S+1)+L'

        READ (HQ')                | the tower heights
        READ (CQ')                | their costs
        READ(FQ')                 | the base widths
        FQ'=0.5FQ'
```

```
JUMP2
PRINTLINE
TOWER COST IS SUSPECT
PRINT(C(Q'))3,0
2)REPEAT

        REPEAT

        J=S+1

        RETURN
        **



ROUTINE 33
>> MAG TAPE DATA TRANSFER ROUTINE
C=C+1
T'=0(1)1
READFWD(1,D(100+512T'))          | reads the test sites and their elevations
REPEAT                           | to the buffer


PRINTLINE
TEST SITE RECORDS


N=0

51)T=N/2
VT=D(100+N) ?
XT=ØINTPT(D(101+N))?             | sets up test sites at XN and VN
GT=1000*(D(101+N)-XT)            | drop in 80 FT SQUARE
JUMP1,GT<100
GT=-(-100 + GT)
1)GT=GT?
JUMP52,XT=-2
N=  N + 2
JUMP51

52)N=N/2 -1?
NEWLINE
NEWLINE
PRINTLINE
CLEARANCE RECORDS

T'= 0(1)4
READFWD(1,D(100 +512T'))         | reads the clearance elevations to the buffer
REPEAT
```

```
R= 0
53) JUMP54,D(R +100)=\\\-1
AR=D(R +100) ?   | transfers the details of the clearance elevations
R= R + 1
JUMP53

54) AR= \\-1
JUMP55,XR=A(R-1)
PRINTLINE
ERROR IN TEST SITE AND CLEARANCE DATA
END


55) T=R+1
R'=R+5
L=T(1)R'
AL=D(100 + L)
REPEAT
B5=ØINTPT(.01*D(104+R) + .01)
T'=ØINTPT(D(104+R) - 100B5 + .001)
\\\B7=T'
        K=ØINTPT((D(105+R)*.01) + .0001)        | the starting angle of the section
B\\10=K
H'= D(105+R)-100K\\      | the section no
F'=D(101+R)      |\\ from
E'=D(102+R)      | limit
G'= D(103+R)     | to
PRINTLINE
SECTION NO
PRINT(H')2,0
CAPTION
  SPAN LIMIT
PRINT(E')3,0
CAPTION
FEET FROM
PRINT(F')3,0
CAPTION
  TO
PRINT(G')3,0
CAPTION
  FEET OF LINE DISTANCE
NEWLINE
PRINTLINE
PREV SECTION L-
PRINT(B4)1,0
CAPTION
TYPE  // CURRENT SECTION L-
PRINT(B5)1\\,0
CAPTION
TYPE // NEXT SECTION L-
PRINT(B10)2,0
PRINT('TYPE')

newline
printline
starting angle=
```

```
PRINT(B6)3,0
CAPTION
FINISHING ANGLE=
PRINT(B7)3,0
NEWLINE
PRINTLINE
MAG TAPE TRANSFER UPTO BLOCK
PRINT(7C + 1)3,0
CAPTION
ON MAGTAPE

| this block is to compute anglo of turn indicos

JUMPDOWN32
B12=T'
T'=ØINTPT(B6+.01)
JUMPDOWN32
B11=T'
PRINTLINE
STARTING ANGLE TOWER INDEX=
PRINT(B11)2,0
CAPTION
FINISHING ANGLE TOWER INDEX=
PRINT(B12)2,0



NEWLINE
NEWLINE
\\RETURN


32)JUMP33,T'>10
        T'=1
        JUMP40
33)JUMP34,T'>30
        T'=2
        JUMP40
34)JUMP35,T'>\\\60
        T'=3
        JUMP40
35)JUMP36,T'>90
        T'=4
        JUMP40
36)JUMP37,T'>95
        T'=5
        JUMP40
37)T'=6

40)RETURN
**
```

```
ROUTINE 34
>> TRANSFORMATION ROUTINE NOTE THIS ROUTINE OPERATES ON THE CLEARANCE DATA
>> AND TRANSFORMS THEM TO ELEVATION AND INCLINATION PARAMETERS FOR EACH LINE SECTIOI


R= -1
1) R= R + 1
JUMP7, A(5R) = -1
JUMP6, A(5R + 3) > 100

JUMP2, A(5R + 1) = AØMAX(A0,5R,5R+2)



A(5R) = .5(AØMAX(A0,5R,5R+2) -A(5R+1)) + A(5R+1)  |height of a point on the ground e
| under the line                                                                elevation

A(5R+1) =( A(5R))/20-(A(5R+1))/20 ?
A(5R+1) = ØARCTAN(A(5R+1)) ?
JUMP1


2) A(5R) = A(5R+1) ?  | this is for level ground and ridges
A(5R + 1) = 0 ?
JUMP1




6) A(5R+3) = A(5R+3) - 100 ?
JUMP6,A(5R+3) > 100
A(5R) = -10
JUMP1


7)RETURN

**
ROUTINE 35
| SPAN SWOPING ROUTINE FOR SECTIONS WITH RESTRICTED SPAN


JUMP19,Z'>0.1
JUMP19,XL-F'<0.1


Y'=B2
B2=E'
Z'=1
PRINTLINE
NEW SPAN LIMIT=
PRINT\\(B2)4,0
CAPTION
FEET    FROM LOCATION
PRINT(XL)5,0
CAPTION
FEET
```

```
19)JUMP20\\,Z'>1.1
JUMP20,XL<F'+Y'
JUMP21,(F'+Y')<G'

30)B2=Y'
Z'=2
PRINTLINE
SPAN LIMIT SETBACK AGAIN AT LOCATION=
PRINT(XL)5,0
20)RETURN


21)JUMP20,XL<G'
JUMP30

**



ROUTINE 36
|routine to initialise the indicator(state) parameters

Z'=0
W3=0
L=1
\\\\\W4=0                    |line type parameter initialised
JUMP81,B5>2
W4=1
JUMP82
81)W3=1                      |it is an 16 type line
82)D'=0
J=10(1)13
WJ=0
REPEAT


RETURN
**
```

```
ROUTINE 39
| ROUTINE TO COMPUTE THE SOLUTION VECTOR Y(61-64) , I'


W1=1
W2=1


JUMP1,L'=0
W2=0
1) JUMP2,L=N
W1=0


2)JUMP3,(W1+W2)<0.9
12)JUMP13,B4>2
W6=1
JUMP14

13)W5=1
14)JUMP15,B10>2
W8=1
JUMP16
15)W7=1


16)WO=S'+W2*(S+1)*ØINTPT(B11+.01)+W4W6W2*72 + 72*(W4-W2)*W4
K= ØINTPT(WO + .01)
WO=T+W1*(S+1)*ØINTPT(B12+.01)+72*W4W1W8+72W4(W4-W1)
T'=ØINTPT(WO+.01)

Y61=VL' + HK - GL'*(.025)*FK
Y62=VL + HT' - GL*(.025)*FT'
Y63=Y61 + (1-W2)(W3*Y65 + W4*Y66)(1-W12)
Y64=Y62 +(1-W1)(W3*Y65 +W4*Y66)(1-W13)


W9=K


K=5(1)8
WK=0
REPEAT

K=ØINTPT(W9 +.01)


101)I'=K' +C(T')

RETURN
```

```
3)JUMP4,R5>2
Y61=VL' + H(S' +72) - GL'*(.025)*F(72+S')
Y62=VL +H(T+72) -GL*(.025)*F(72+T)



Y63=Y61+Y66(1-W12)
Y64=Y62+Y66(1-W13)


T'=T+72
K=S'+72


JUMP101



4)Y61=VL' + H(S') -GL'*(.025)*F(S')
Y62=VL + H(T) - GL*(.025)*F(T)
Y63=Y61 + Y65(1-W12)

Y64=Y62 + Y65(1-W13)

T'=T
K=S'
JUMP101
**

ROUTINE41
| this routine is for end result swop and cost level setting
M=N(S+1)
L=O(1)S
M'=M+L
UNPACK(DM',O,K')
JUMP1,K'=0
X'=K'+X'-1       | the step increase in cost level
P=K'-1
JUMP2
1)REPEAT

2)M=N(S+1)
PRINTLINE
THE ACCUMULATED COST=
PRINT(X')5,0
L=O(1)S
M'= M + L
UNPACK(DM',O,K')
JUMP8,K'=0
K'=K'-P
8)P'=2000
S'=0
UNPACK(DM',3,T)
L'=0
R=1
M'=L
V'=N
N=M'
JUMPDOWN(R5)
\\N=ØINTPT(V'+0.1)
NEWLI
```

```
NEWLINE3
PRINTLINE
THE COST
PRINT(K')4,0
CAPTION
      HEIGHT INDEX OF TOWER
SPACE4
PRINT(T)3,0

REPEAT


B4=B5

B4=B5
B6=B7
RETURN


**


ROUTINE 51
| ROUTINE TO SEARCH BACK AND PRINT THE SOLUTION
P=0(1)S
L=N
M'=(N)*(S+1) + P
J=M'
JUMPDOWN(R8)
JUMP3,K'=0
PRINT(M')5,0
PRINT(J)5,0
PRINT(L')3,0
W13=0
W12=0
W12=1
JUMP9,S'>S
W12=0
9)JUMPDOWN2
JUMP3,K'=0
JUMP4

3)PRINTLINE
UNECONOMIC SOLUTION UPTO THE PRESENT TOWER TYPE/
PRINT(P)3,0
4)REPEAT

RETURN
```

```
1)L=L'
JUMPDOWN(R8)
JUMP21,T>S
W13=0
JUMP22
21)W13=1
22)JUMP23,S'>S
W12=0
JUMP2
23)W12=1


2)JUMPDOWN(R39)
NEWLINE
PRINT(S')4,0
PRINT(K)4,0
PRINT(T)4,0
PRINT(T')4,0
PRINT(L)4,0
PRINT(L')4,0
PRINTLINE
S'  k  t   tttt'    l   L' in order
\\I'=K'
PRINTLINE
COST UPTO THIS POINT=
PRINT((I'+X'))5,0
CAPTION
COST OF THE CURRENT TOWER=
PRINT(C(T'))3,0
CAPTION
IN PERCENT OF THE LOWEST STANDARD TOWER
JUMPDOWN(R2/90)


JUMP10,S'>S
M'=(S+1)L' +S'
J=M'

JUMP1,M'≥(S+1)
JUMP11


10)J=(S+1)L' +S' -S-1
M'=J+5000
JUMP1


11)PRINTLINE
PRESENT TOWER TYPE AT START =
print(S')2,0
CAPTION
ABSOLUTE TOWER TYPE=
PRINT(K)3,0
PRINTLINE
BEGINNING OF LINE SECTION REACHED
NEWLINE 4
RETURN
**
```

```
TITLE
OPTIMUM TOWER SPACING WITH SUBOPTIMALS / NITRA
CHAPTER 0

A->2560
U->20
X->512
V->512
D->10000
E->5000
;->60
Z->10080
B->19
C->160
H->160
Y->70
F->170
G->512
W->20



SELECT INPUT2
READ(O')
READ(X')
READ(D)
SELECT INPUT1


WIND TAPE(1,8')
8'=0
JUMPDOWN(R31)           | input routine
JUMPDOWN(R32)                 |block to generate starting values
50)JUMPDOWN(R33)        |brings down the data from tape
JUMPDOWN(R34)           |processes the data
JUMPDOWN(R36)           |initialises for the section


51)PRINT(L)3,0
\\
JUMP52,L≠N              |loop entry p\\oint within the section
W11=0
52)U'=0
R=-1
JUMP\\61,Z'>0.1
JUMP20,E'=B2            |checks for current span setting
61)JUMPDOWN(R35)              |checks the current span


|this block\\ computes the band of search
```

```
20)R=R+1
JUMP20,A(5R+4)<XL
L'=L

21)L'=L'-1
JUMP21,\\\\(XL-XL'+.01)<B1
PRINT((XL-XL'))4,5
PRINT(L')2,0
PRINT(B1)\\\\4,0
P=L'
22)L'=L'-1
JUMP23,L'=-1
JUMP22,(XL-XL')<B2
23)I=L'+1
PRINT(I)2,0
T=-1


89)JUMP90,W11>0.1
W13=0

T=0(1)S                    |loop for optimal solutions
JUMP24
91)W13=1
JUMP24

90)T=T+1
JUMP91,T>S
W13=0


24)M=250000                |very high cost
Q'=0                       |to test the solution
L'=I(1)P                   |the band of feasible connection
JUMP29,W'<0.1
JUMP27,(L'-D')>-0.1
29)W10=0
JUMP28
27)W10=1
28)M'=(S+1)L'-1
S'=-1


45)JUMP46,W10>0.1
S'=0(1)S
JUMP25
46)S'=S'+1

JUMP25,S'≤S
J=J+1
W12=1
JUMP47,S'≠S+1
J=L'(S+1)

47)M'=J+5000
JUMP26

25)W12=0
M'=M'+1
J=M'
26)JUMPDOWN(R4)            |bring down the solution
```

```
JUMP30,K'≤0
Z=XL-XD'
JUMPDOWN(R39)              |compute the solution vector
JUMP32,I'>M
JUMP32,I'<0                |eliminates the imaginary towers
JUMP30,Z>B3                |double span restriction
JUMPDOWN(R1)


\\;1=;3
;2=P'
\\JUMP62,L'=0
JUMP30,(;1+;2)<0.35Z       |weight restriction
62)JUMPDOWN(R2)

JUMP31,Q=0                 |linkage successful
30)JUMP37,W10>0.1
REPEAT
JUMP32


37)JUMP32,S'=(2S+1)
JUMP45
31)Q'=1


U'=1
JUMP39,T≤S
M=I'
J'=\\3(T-S-1)
\\Z(T-S-1)=I'
YJ'=S'
J'=J'+1
YJ'=20000
J'=J'+1
YJ'=L'
JUMP32


39)M=I'
J'=3T
YJ'=S'
ZT=I'
J'=J'+1
YJ'=XL-XL'-;1
J'=J'+1
YJ'=L'
```

```
32)REPEAT
JUMPDOWN(R10)
40)JUMP41,W11>0.1
REPEAT
JUMP43



41)JUMP42,T≠S              |optimal store for T≤S
JUMPDOWN(R9)
JUMP89


42)JUMP89,T<2S+1
43)JUMPDOWN(R9)            |optimal and suboptimal storage


JUMP60,U'<0.1              |no solution exists
44)L=L+1
JUMP51,L≤N

PRINTLINE
BLOCK TO SEARCH BACK AND PRINT SOLUTION
JUMPDOWN(R51)

JUMPDOWN(R41)
JUMP50,C<D
END

60)JUMPDOWN(R11)
JUMP51
CLOSE


***Z
```

APPENDIX 8

```
JOB
[illegible]
INPUT
0 DISK
1 TAPE[illegible]
OUTPUT
0 LINEPRINTER 600 LINES
1 LINEPRINTER 600 LINES
STORE 29/35 BLOCKS
COMPUTING 20000 INSTRUCTIONS
***C

COMPILER ALGOL
DISK

begin integer m,n,D,row;
      boolean op,np; integer ra,rb,rc,rd;real re,rol;
real procedure random;
      begin rd:=5*rd;
if rd>rc then rd:=rd-rc;
if rd>rb then rd:=rd-rb;
if rd>ra then rd:=rd-ra;random:=rol*rd;
end random;


real procedure instr ;
      begin code
1136,      0,      0,      0;
355,  0,      -,      instr;
      end of instruction ctr extract;

integer procedure primlex(a,l,re,m,n,D);integer array a;integer l,m,n,D;real re;
                                                                  re;

      begin integer array row,list[1:n+m+1];boolean bul;integer i,k,j,p;
      real rol,prel,ex;
      list[1]:=1;for i:=2 step 1 until n+m+1 do
                        if a[i,1]>0 then
                  begin if a[i,1]/a[i,1]=re then
                        begin list[1]:=list[1]+1;list[list[1]]:=i
                        end;
                  end;
      i:=n+1;
      loop:   i:=i+1;p:=0;
      for j:=1 step 1 until n+1 do
            if a[i,j]≠0 then
                        begin
                                          p:=p+1;k:=j;if p>1 then goto yes
                        end;
for j:=1 step 1 until n+m+1 do row[j]:=a[j,k]; goto ratio;
yes:  for j:=1 step 1 until n+m+1 do row[j]:=if i=j then D else 0;
```

```
        ratio:for j:=1 step 1 until n+m+1 do print(list[j],3,0);
                for j:=1 step 1 until n+m+1 do print(row[j],3,0);writetext(<mntr,
                                                                          );
        newline(2);
                rel:=9999999999;bul:=false;for j:=2step 1 until list[1] do
                begin ex:=row[list[j]]/a[list[j],1];
                   ifex-0.0000000001<rel then
                                                    begin prel:=rel;rel:=ex;
                        primlex:=list[j];bul:=true
                                                  end;
                                ifbul and prel=rel
                        then begin  p:=1;forj:=2 step 1 until list[1] do
                                if row[list[j]]/a[list[j],1]=rel
                                 then begin p:=p+1;list[p]:=list[j];
                                                  end;list[1]:=p;ifi<n+m+1 then
                                                                 goto loop;
                                i:=1\\\\;goto loop;
                                end;
                end;

end of primlex;
integer procedure rem(pi,q,div);valuepi,q;integer pi,q,\\\\div;
        beginifpi<0thenpi:=-pi;ifq<0thenq:=-q;
        div:=pi/q-0.5;rem:=pi-q*div;
        endrem;

        procedure diagonal(ar,n,old); integer array ar;
        integer n,old;
begin integer  i,j; for i:=2 step 1 until n do
           for j:=2 step 1 until n do
           if i=j then  ar[i,j]:=old;
end;
procedure input(a,m,n,D,cpr,npr,more);
        integer m,n,D; integer array a;
                boolean cpr,npr,more;
        begin integer i,j;
        fori:= 1 step 1 until n + 1 do
        forj:= 1step 1 until n + 1 do
                        a[i,j]:=0;
                 for i:=1, n+2 step 1 until n+m+1 do
              for j:=1 step 1 until n+1 do
                a[i,j]:=read; D:=read;
                cpr:=readBoolean;
                npr:=readBoolean;
                more:=readBoolean;
        end input;
```

```
        procedure monitor(a,m,n,D,it);
        integer m,n,D,it; integer array a;
                begin integer i,j;

                        newline(1); writetext(←iterno=→); print(it,8,0);
                        newline(1); writetext(←denom=→); print(D,8,0);
                        newline(2); writetext(←tableau→);
                        newline(1);
                        for i:=1 step 1 until m do begin newline(2);
                        for j:=1 step 1 until n do print(a[i,j],8,0);
                                end;
        loop:           end monitor;

integer procedure hcf(a,b);value a,b;integer a,b;
                begin integer s;
        loop:   if b>a then  begin s:=b;b:=a;a:=send;
                s:=a/b-0.5;a:=a-b*s;
                ifa=1 then begin hcf:=1;gotoout; end;
                ifa>1 then goto loop;
                hcf:=b;
out:    end of hcf;

                procedure  primalduallexico(a,m,n,D,cpr,npr,nobound,
                        infeasible,exit);
                        value m,n,cpr,npr; integer m,n,D; integer array a;
                        boolean cpr,npr;
                        label nobound,infeasible,exit;

                        comment accelerated gomory employs gomory's cutting plane
                                                        technique
                        with better cuts;

                        begin boolean primal,dual,mk;
                        integer i,j,lexico,k,l,r,iter,p,q;
                        real re,pre,ex;
                        integer array lis2[1:n+m+2], list[1:n+m+3];
        procedure pivot(a,D,r,l,iter,cr,nr);
                integer D,r,l,iter;
                integer array a;
                boolean cr,nr;
                        begin integer i,j,k,t;
                                iter:=iter+1; k:=a[r,l];  t:=sign(a[r,l]);
                                for j:=1 step 1 until n+1 do if j≠l then
                                begin
                                                for i:=1 step 1 until n+m+2 do
        ifi≠rthen                               a[i,j]:=t*(a[i,j]*k-a[i,l]*a[r,j
                                end;
                        D:=abs(k); for i:=1 step 1 until m+n+2 do a[i,l]:=-t*a[i
                                                        l];
        forj:= 1step1until1n + 1 doifj≠l thena[r,j]:=0;
                        comment the following coding is for monitor purpose;
                        if cr and nr then monitor (a,m+n+2, n+1,D,iter)else
```

```
                                    begin if
                                                    cr and not (primal and dual) then
                                            monitor(a,m+n+2,n+1,D,iter)
                                    end;
                            end pivot;
primal:=dual:=false;iter:=0;

LP1:mk:=false;
comment the following tests for the dual feasibility;
l:=2; for i:= 3 step 1 until n+1 do if a[1,i]<a[1,l]then l:=i;
        ifa[1,l]<0then
begin comment then primal l*simplex method now follows;

        pre:=re:=9999999999;
                for i:= 2 step 1 until m+n+1 do
                begin ifa[i\\,l]>0 then
                        begin
                           ex:=a[i,1]/a[i,l];
                            ifex-0.000000000001<rothen
                                                begin
                                                   pre:=re;re:=ex;k:=i;mk:=true
                                                end;
                        end;
        end;

                ifpre=re and mk then
                                k:=primlex(a,l,ro,m,n,D);

if not mk then goto nobound;
        writetext(←primalslexicoskl→);print(lexico,3,0);print(k,2,0);print(l,3,
        newline(2);
                                pivot(a,D,k,l,iter,npr,cpr);
                        goto LP1;
end;

dual:=true;
L2:     fori:=2 step 1 until n+m+1 do if a[i,1]<0 then
                                                                begin primal:=f
                                                            FALsefalse;k:=i;gotoLP2
                                                end;
                primal:= true;
        ifprimal and dualthengoto int elsegoto LP1;
LP2: comment the dual l-simplex method now follows;
        mk:=false;lexico:=1;for j:=2 step 1 until n+1 dolist[j]:=j;list[1]:=n+1;
        dual\\visit:     pre:=re:=99999999999;
                forj:=2 step 1 until list[1] do
                         ifa[k,list[j]]<0 then
                                begin
                                   ex:=abs(a[lexico,list[j]]/a[k,list[j]]);
                                    if ex-0.00000000001<rothen
                                          begin pre:=re;re:=ex;l:=list[j];mk:=tru
                                          end;
                                end;
                        ifpre=re and mk then
                                begin dual:=false; lexico:=lexico+1;p:=1;
                                   forj:=2 step1 untillist[1] do ifa[k,list[j]]<
                                                                0then
```

```
                                begin     ifabs(a[lexico-1,list[j]]/a[k,list[j]])
                                                                        =ro
                                          then
                                             begin p:=p+1;
                                                 list[p]:=list[j];
                                            end;
                                end;
                                list[1]:=p;
                                goto dualvisit;
                                end;

                        if not mk then goto infeasible;
                        writetext(←dual$lexico$k,l→);print(lexico,3,0);print(k,3,
        print(1,2,0);newline(2);
                        pivot(a,D,k,l,iter,cpr,npr); goto L2;

        int: monitor(a,m+n+2,n+1,D,iter);goto exit;
end of primalduallexico;
ra:=34359738368;rc:=137438953472;rb:=68719476736;rd:=13543288579;rol:=1/ra;

        selectinput(1);
again: m:=read; n:=read;
                        begin integer array a[1:m+n+3, 1:n+1]; booleanmore;intego
                        tot:=instr/2048;
                                input\\(a,m,n,D,cp,np,more); diagonal(a,n+1,-D);
                                select output(1);
newline(1);
                                monitor(a,m+n+2,n+1,D,0);
                                primalduallexico(a,m,n,D,cp,np,penthouse,soho,hom
        penthouse:writetext(←solution unbounded→); goto st;
                soho:writetext(←the$PROPOSITION$infeasible→);
                        goto st;
                homo:writetext(←last$tableau$was$CONTINUOUS$optimal→);

BLOK:begin integer array S,b,Fr,left,val,hcfs,bound[0\\:n+1];
        integer i,comdiv,rems,j,k,l,q,pt,flpt,rhs,r,p,t;
        boolean mark;boolean arraybreq[0:n+1];
        fori:=0step 1 until n+1 do
        beginS[i]:=b[i]:=Fr[i]:=left[i]:=val[i]:=hcfs[i]:=bound[i]:=0;
                br\\eq[i]:=false
        end of initialisation;
        i:=j:=k:=l:=q:=pt:=flpt:=rhs:=r:=p:=t\\\\\\:=0;
        mark:=false;
        forj:=2 step 1 until n+1 do Fr[j]:=a[1,j];
```

```
        Fr[1]:=if a[1,1]<0 then D-rem(abs(a[1,1]),D,q)else rem(a[1,1],D,q);
        for j:=2 step 1 until n+1 do if Fr[j]=0 then
                begin writetext(‹dualdegeneracy›);goto st
                end;


comment the following orders the coeffecients;
        S[2]:=2;
        for p:=3 step 1 until n+1 do
        begin if Fr[p]=0 then goto lexlist; q:=2;
                loop: if Fr[p]<Fr[S[q]]then
                        begin if S[q]=2 then goto out;q:=S[q];goto loop
                        end;
                out:t:=S[q];S[q]:=p;S[p]:=t;
        goto jump;
        lexlist:comment lexlist will be coded here;
   jump:end;
        q:=t:=2;if Fr[2]=0 then
        begin for t:=t while S[t]≠2 do t:=S[t];
                S[t]:=S[S[t]];q:=t
        end;
        if Fr[2]>Fr[S[2]]then
                begin for t:=t while S[t]≠2 do t:=S[t];q:=t
                end;


comment the following orders the coeffecients by backward reference;
        t:=q;
        rep: b[S[t]]:=t;t:=S[t];if t≠q thengoto rep;
comment the following produces the local group hcfs;
        hcfs[q]:=Fr[q];t:=q;
        for t:=b[t]while t≠q do hcfs[t]:=hcf(Fr[t],hcfs[S[t]]);
comdiv:=hcfs[S[q]];for i:=2 step 1 until n+1 do
        begin Fr[i]:=Fr[i]/comdiv+0.5;hcfs[i]:=hcfs[i]/comdiv;
        end;
        comment the bound computation comes here but we skip it for the time being

t:=instr/2048;
r:=1;
OUT:newline(3);
        writetext(‹head of list›);print(q,3,0);newline(1);
        writetext(‹order of forward›);for i:=0 step 1 until n+1 do
        print(S[i],4,0);newline(1);
        writetext(‹b order›);for i:=0 step 1 until n+1 doprint(b[i],3,0);newline(1)
        writetext(‹hcfs›);for i:=0 step 1 until n+1 doprint(hcfs[i],4,0);newline(
```

```
comment the actual search now follows;
search1:flpt:=pt:=S[q];
        left[flpt]:=(Fr[1]+D*r\\\\)/\\\\\\\\ comdiv;print(left[flpt],9,0);writetext(←i

        search2:ifhcfs[flpt]≠1then goto god;
                carryon:val[flpt]:=left[flpt]/Fr[flpt]-0.5;
                        rhs:=left[flpt]-Fr[flpt]*val[flpt];
                onward:flpt:=S[flpt];
                bounder:ifbreq[flpt]then beginifval[flpt]>bound[flpt]then

                                                goto stepback
                                                end;
                                        goto test;
        god:i:=left[flpt]/hcfs[flpt]-0.5;
                rems:=left[flpt]-hcfs[flpt]*i;
                if rems=0 then goto carryon;
        ifflpt≠q then goto stepdown;

        ifpt≠q then goto stepdown;
        if mark thengotosa\\\\\;r:=r+1;writetext(←cycle→);print(r,3,0);newline(1);
        writetext(←instr=→);print(instr/2048-t,4,0);t:=instr/2048;

        goto search1;

test:if rhs=0 then goto solution;
        left[flpt]:=rhs;
        goto search2;

solution: i:=b[flpt];fori:=S[i]while i≠S[\\\\q]doval[i]:=0;


                \\\\ifinstr/2048<19900 then goto stepback\\\\\;
        newline(1);writetext(←solutionvector→);
        fori:=0step 1 untiln+1 do print(val[i],4,0);
        writetext(←rhs→);print(left[flpt],9,0);
mark:=true;

        stepback:flpt:=b[flpt];
                stepdown:flpt:=b[flpt];
        ifval[flpt]=0 then goto stepdown;val\\[flpt]:=val[flpt]-1;
        rhs:=left[S[flpt]]+Fr[flpt];
        if val[flpt]≠0 orflpt≠pt then goto o\\nward;
        pt:=S[pt]; goto onward;
end of BLOK;
                st:newline(1);writetext(←countsend→);print(instr/2048-tot,5,0);r

 if more then goto againend;

end of program;
***C
```

DATA
TABLEAU
15 15
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
-8 -1 0 0 0 -1 -1 -1 0 0 0 -1 -1 -1 0 -1
-8 0 -1 0 0 -1 0 0 -1 -1 0 -1 -1 0 -1 -1
-8 0 0 -1 0 0 -1 0 -1 0 -1 -1 0 -1 -1 -1
-8 0 0 0 -1 0 0 -1 0 -1 -1 0 -1 -1 -1 -1
-7 -1 -1 0 0 0 -1 -1 -1 -1 0 0 0 -1 -1 0
-7 -1 0 -1 0 -1 0 -1 -1 0 -1 0 -1 0 -1 0
-7 -1 0 0 -1 -1 -1 0 0 -1 -1 -1 0 0 -1 0
-7 0 -1 -1 0 -1 -1 0 0 -1 -1 0 -1 -1 0 0
-7 0 -1 0 -1 -1 0 -1 -1 0 -1 -1 0 -1 0 0
-7 0 0 -1 -1 0 -1 -1 -1 -1 0 -1 -1 0 0 0
-6 -1 -1 -1 0 0 0 -1 0 -1 -1 -1 0 0 0 -1
-6 -1 -1 0 -1 0 -1 0 -1 0 -1 0 -1 0 0 -1
-6 -1 0 -1 -1 -1 0 0 -1 -1 0 0 0 -1 0 -1
-6 0 -1 -1 -1 -1 -1 -1 0 0 0 0 0 0 -1 -1
-5 -1 -1 -1 -1 0 0 0 0 0 0 -1 -1 -1 -1 0
1
false **true** **false**

5 5
0 0 0 0 0 1
-506 57 -38 18 -4 -48
-721 -49 -27 49 -88 -31
-569 38 94 -16 -18 -20
-1784 -59 -90 -93 72 80
-1771 -17 -2 -27 88 -88
1
false **false** **false**
***Z

APPENDIX 9

```
COMPILER ALGOL
SYM-9 PROG


begin real ro;
        boolean primal,dual,Tst,mk;
        integer m,n,no,tot,t,iter,Phno,ch1,ch2,outgoing,incoming,compi,ra,rb,rc,
                                                                    rd;


real procedure instr;
        begin code
        1136,0,0,0;
        356,0,-,instr;
        end of instr;

real procedure random;
        begin
        rd:=5*rd; if rd>rc then rd:=rd-rc;
                if rd>rb then rd:=rd-rb;
                if rd>ra then rd:=rd-ra;
                random:=ro*rd;
end random;



integerprocedureroadch;
begin code
 0121,79,127,1;
 0101,127,0,2;
 0124,80,80,0;
 1524,0,80,0;
 0320,0,0,*140000070;
 0356,0,-,roadch;
endroadch;

procedureprintch(i);
valuei;integeri;
begin code
 1300,80,-,i;
 0172,80,0,130;
 0121,79,0,0;
 0226,79,0,*4;
 0163,80,79,0;
 0121,79,127,1;
 0101,127,0,0.4;
 0121,0,0,0;
endprintch;
```

```
procedure monitor(a,it,k,limit,vcol,vrow,Pt);value it,k,limit;
        real array a;
        integer array vrow,vcol,Pt;
        integer it,k,limit;

        begin integer i,line,j;

                writetext(‹iter#no=›);print(it,4,0);
        line:=entier(it/5+0.0001);if line*5=it then
         newline(1);
        \\\\
if not primal or not dual then goto fin;
if limit>2 then goto extra;
for i:=0 step 1 until limit do
        begin  printch(vcol[i]);print(i,2,0);space(\\10);
        end;
                for i:=0 step 1 until limit do
        begin
                newline(1);printch(vrow[i]);print(i,2,0);
                for j:=0 step 1 until i do print(a[Pt[i]+j],4,8);
                if i=k then   writetext(‹<--›);
        end;
        extra:  newline(6);line:=0;
         for i:=0 step 1 until limit do

        begin printch(vrow[i]);print(i,2,0); writetext(‹=›);
                print(a[Pt[i]+0],4,5);line:=line+1;
        if line=5then begin newline(2);line:=0 end;

        end;    newline(4);writetext(‹function#value=›);print(a[0]/2,6,8);


\\
fin:    end of monitor;

procedure input(a,vcol,vrow,m,n,Pt)\\\\;value m,n;
        integer m,n;real array a;
        integer array vcol,vrow,Pt;
        begin integer i,j;
comment procedure input \\reads and sets up the starting state and
        tableau.The next bit sets up the one dimensional symmetric matrix;
        Pt[0]:=0;
        for i:=1 step 1 until n+m doPt[i]:=Pt[i-1]+i;
comment the objective and the p-vector follows;
        a[Pt[0]+0]:=0;
        for i:=1 step 1  until n do
        begin a[Pt[i]+0]:=read;a\\[Pt[i]+0]:=-a[Pt[i]+0]\\\\;print(a[Pt[i]+0],5,5)\\
        end;
        for i:=1 step 1 until n do
        for j:=1 step 1 until i do
                                a[Pt[i]+j]:=read;
comment the above assumes C isprosented with a negative sign;
```

```
        for i:= n+1 step 1 until n+m do
        for j:=0 step 1 until i do
                if j< n then begin a[Pt[i]+j]:=read ;print(a[Pt[i]+j],3,5)end
         else a[Pt[i]+j]:=0;

        for vcol[0]:=readch while vcol[0]=1 or vcol[0]=123 do;
        forvrow[0]:=readch while vrow[0]=1 or vrow[0]=123 do;

        for i:=0 step 1 until n+m do
        begin if i<n then
                        begin
                        vrow[i]:=vrow[0];vcol[i]:=vcol[0]
                        end
                else
                        begin vcol[i]:=vrow[0];vrow[i]:=vcol[0]
                        end
        end;
end of input;


procedure pivot(a,k,limit,Pt,itor,vcol,vrow);value limit,k;
        real array a;
        integer array vcol,vrow,Pt;
        integer  limit,iter,k;
        begin integer i,j,sgn; real Div,rowmplr;
        Div:=1/a[Pt[k]+k];
        for i:=0 step 1 until k-1 do
        beginsgn:=ifvcol[k]≠vrow[i] then 1 else -1 ;

        rowmplr:=a[Pt[k]+i]*Div;

        for j:= 0 step 1  until i do
                a[Pt[i]+j]:=a[Pt[i]+j]-sgn*(a[Pt[k]+j]*rowmplr)
        end;

        fori:=k+1 step 1  until limit do

begin   rowmplr:=a[Pt[i]+k]*Div;
        for j:=i step -1  until 0 do
        begin

         if j>k then begin sgn:=if vcol[j]≠vrow[k]then 1 else -1;
                a[Pt[i]+j]:=a[Pt[i]+j]-sgn*a[Pt[j]+k]*rowmplr
                                end;
                if j<k then a[Pt[i]+j]:=a[Pt[i]+j]-a[Pt[k]+j]*rowmplr
        end
end;
```

```
        for i:=0 step 1 until limit do
                begin
        if i<k then a[Pt[k]+i]:=a[Pt[k]+i]*Div;
        if i>k then a[Pt[i]+k]:=-a[Pt[i]+k]*Div
                end;
        a[Pt[k]+k]:=1/a[Pt[k]+k];

        sgn:=vrow[k];vrow[k]:=vcol[k];vcol[k]:=sgn;
        iter:=iter+1;
        end of pivot;
procedure duopivot(a,in,out,lmt,Pt,iter,vcol,vrow);value in,out,lmt;
        integer in,out,lmt,iter;integer array vcol,vrow,Pt;
        real array a;
        begin real array rhi,rlo,chi,clo,srhi,srlo,schi,slo[0:lmt];
        integer i,j,lo,hi;real inversehilo,inverselohi,mplr1,mplr2;
        lo:=in;hi:=out;if lo>hi then beginlo:=hi;hi:=in end;
        for i:=0 step 1 until lmt do
begin
        if i<lo then beginrlo[i]:=a[Pt[lo]+i];rhi[i]:=a[Pt[hi]+i];
                clo[i]:=ifvrow[i]=vcol[lo]then-a[Pt[lo]+i]elsea[Pt[lo]+i];
                chi[i]:=ifvrow[i]=vcol[hi]then-a[Pt[hi]+i]elsea[Pt[hi]+i];
                        end;
        if i>lo andi< hi then begin
                rlo[i]:=ifvcol[lo]=vrow[i]then -a[Pt[i]+lo]elsea[Pt[i]+lo];
                rhi[i]:=a[Pt[hi]+i];clo[i]:=a[Pt[i]+lo];
                chi[i]:=ifvcol[hi]=vrow[i]then-a[Pt[hi]+i]elsea[Pt[hi]+i];
                                end;

        if i> hi then begin rlo[i]:=ifvcol[lo]=vrow[i]then-a[Pt[i]+lo]
                                                elsea[Pt[i]+lo];
                rhi[i]:=ifvcol[hi]=vrow[i]then-a[Pt[i]+hi] else a[Pt[i]+hi];
                clo[i]:=a[Pt[i]+lo];
                chi[i]:=a[Pt[i]+hi];
                        end;
end;

        inversehilo:=1/rhi[lo];inverselohi:=1/clo[hi];
for i:=0 step 1 until lmt do
beginrlo[i]:=rlo[i]-rhi[i]*clo[lo]*inversehilo;
        srhi[i]:=rhi[i]*inversehilo;
        slo[i]:=-clo[i]*inverselohi;
        schi[i]:=chi[i]-clo[i]*rhi[hi]*inversehilo
end;
        srlo[lo]:=slo[lo];schi[hi]:=srhi[hi];
        srhi[lo]:=slo[hi]:=1/rhi[lo];
```

```
inversolohi:=1/schi[lo];
for j:=0 step 1 until lmt do
        if j≠hi or j≠ lo then

begin

        mplr1:=rhi[j]*inversohilo;mplr2:=srlo[j]*inversolohi;
                for i:= j step 1 until lmt do if i≠hi or i≠lo
        then
        a[Pt[i]+j]:=a[Pt[i]+j]-clo[i]*mplr1-schi[i]*mplr2
end;

        for i:= 0  step 1 until lmt do

begin if i≠ hi then
        rhi[i]:=srhi[i]-srlo[i]*schi[hi]/schi[lo] else
        rhi[i]:=-srhi[i]/schi[lo];

        if i< lo then a[Pt[lo]+i]:=rhi[i] else
        a[Pt[i]+lo]:=if vcol[i]=vrow[lo] then rhi[i] else-rhi[i];

        ifi≠hi then
        rlo[i]:=srlo[i]/srlo[hi]else rlo[i]:=1/srlo[hi];
        if  i< hi then a[Pt[hi]+i]:=rlo[i] else
        a[Pt[i]+hi]:=if vcol[i]=vrow[hi] then  rlo[i] else-rlo[i];
end;
        a[Pt[lo]+lo]:=rhi[hi];a[Pt[hi]+hi]:=rlo[lo];a[Pt[hi]+lo]:=rlo[hi];
i:=vcol[lo];vcol[lo]:=vrow[lo];vrow[lo]:=i;
i:=vcol[hi];vcol[hi]:=vrow[hi];vrow[hi]:=i;
iter:=iter+1;
end of proc duopivot;

integer procedure largest(ch2,a,Pt,n,m,vrow);value n,m,ch2;
        integer ch2,n,m;
        integer array Pt,vrow;real array a;
        begin integer i,j; real stay;

        stay:=a[Pt[0]+0];a[Pt[0]+0]:=0;j:=0;
        for i:= 1 step 1 until  n+m do
        ifvrow[i]=ch2 and a[Pt[i]+0]< a[Pt[j]+0] then j:=i;
        largest:=j;a[Pt[0]+0]:= stay
        end of largest;


integer procedure outgo(a,Pt,ch2,vrow,vcol,in,m,n,unbound,infeas);value ch2,in,
                                                                    m,n;
                integer array vrow,vcol,Pt;
                real array a;
                integer in,m,n,ch2;

        label unbound,infeas;
begin real ratio,nxt,comp;
        integer i,sgn,j;
        integer arraylist[0:m+n];
```

```
ratio:=1000000000;nxt:=0000000000;

        if a[Pt[in]+in]<-0.00000000001 and a[Pt[in]+0]<-0.00000000001
        then begin ratio:=a[Pt[in]+0]/a[Pt[in]+in];j:=in
             end;
        for i:= 1 step 1 until in do
        if a[Pt[in]+i]≠0 and vrow[i]≠ch2 then
begin sgn:=if vcol[i]=vrow[in]then -1 else 1;
        if a[Pt[in]+i]*sgn<-0.00000000001
        then goto skips; if a[Pt[i]+0]<0 then goto skips;
        if a[Pt[i]+0]>-0.000000001 and sgn*a[Pt[in]+i]<0 then goto skips;
        comp:=abs(sgn*a[Pt[i]+0]/a[Pt[in]+i]);
        if comp>-0.000000001 and comp- ratio<0.000000001 then
        begin
        if ratio-comp<0.000000001 thennxt:= ratio;
         j:=i;ratio:=comp
        end;
skips:  end;


        for i:=in step 1 until n+n do
        if a[Pt[i]+in]≠0 and vrow[i]≠ch2 then
begin   if a[Pt[i]+in]<0 then goto skip;

        if a[Pt[i]+0]<0 then goto skips;
        if a[Pt[i]+0]>-0.000000001 and a[Pt[i]+in]<0 then goto skip;
        comp:=abs(a[Pt[i]+0]/a[Pt[i]+in]);
        ifcomp>-0.000000001 and comp-ratio<0.000000001 then
        begin
        if ratio-comp<0.000000001thennxt:= ratio;
        j:=i; ratio:=comp;
        end;
skip:       end;
        outgo:=j;


        if nxt-ratio<0.00000001 then  begin
        list[0]:=0;
        for i:= 1 step 1 until in do
                if a[Pt[in]+i]≠0  and vrow[i]≠ch2 then

        beginsgn:=if vcol[i]=vrow[in]then-1 else 1;
        if a[Pt[i]+0]>-0.000000001 and sgn*a[Pt[in]+i]<0 then goto kips;
```

```
                comp:=sgn*a[Pt[i]+0]/a[Pt[in]+i];comp:=abs(comp);
        if comp>-0.000000001 andcomp-ratio<0.000000001then

                beginlist[0]:=list[0]+1;
                        list[list[0]]:=i;
                end;
kips:   end;

        fori:=m+n step-1 until in+1 do
                ifa[Pt[i]+in]≠0and vrow[i]≠ch2
        then
                begin
                ifa[Pt[i]+0]>-0.000000001 anda[Pt[i]+in]<0 then goto kip;
        comp:=abs(a[Pt[i]+0]/a[Pt[i]+in]);
        ifcomp>-0.000000001and comp-ratio<0.000000001 then
        begin list[0]:=list[0]+1;
                list[list[0]]:=i;
        end;
kip:    end;

        j:=list[0]*random; \\\\\\\ifj=0 thenj:=1;
        outgo:=list[j];
end;

        if ratio>10000000 then begin
                if ch2=vrow[0] then goto unboundelse goto infeas
                                end;
end of procedure;
procedure parametrize(a,m,n,Pt,vcol,vrow,i,Pr,init,an);valuem,n,i,Pr,\\\\an;
                integer m,n,i;
                integer arrayvcol,vrow,Pt; realPr,an;arraya;label init;
        comment an, the original parameter,Pr the new one,i its location
                in the zero column;
begin array b[0:m+n]; boolean basic;real mu,nu,mplr,alpha,beta;
        integer p;
        basic:= false;
        if i<n and vrow[i]≠vcol[0] then basic:= true;
        if basic then goto next;
        if i>n  and vrow[i]=vcol[0] then basic := true;
next:   if basic then

          begin
                nu:=an-\\a[Pt[i]+0\\];nu:=-1000000;
                a[Pt[i]+0]:=a[Pt[i]+0]-an+Pr;goto transfor
          end;
```

```
mplr:=an-Pr;
for p:=0 step  1 until  i do
                b[p]:=if vrow[p]=vcol[i] then-a[Pt[i]+p]
                      else a[Pt[i]+p];

        for p:= i+1 step 1 until  n+n do
        b[p]:=a[Pt[p]+i];
mu:=1000000000; mu:=-1000000000;beta:=a[Pt[i]+0];alpha:=a[Pt[i]+i];

for p:=1 step 1 until  n+n do

        begin
        if b[p]>0  then
                        begin
                                if a[Pt[p]+0]/b[p]<mu
                                then mu:=a[Pt[p]+0]/b[p]
                        end;
        if b[p]<0   then
                        begin
                                if a[Pt[p]+0]/b[p]>mu
                                then mu:=a[Pt[p]+0]/b[p]
                        end;


        a[Pt[p]+0]:=a[Pt[p]+0]-mplr*b[p]
                end;
mu:=an-mu;      mu:=an-mu;
if abs(a[Pt[i]+i])<0.0000000001 then

        begin
                a[Pt[0]+0]:=a[Pt[0]+0]-mplr*2*b[0];goto transfer
        end;
        a[Pt[0]+0]:= if i>n then a[Pt[0]+0]-beta*2*mplr +mplr*mplr*alpha
        else a[Pt[0]+0] +beta*mplr*2 -mplr*mplr*alpha;



transfer:       newline(2);writetext(←critical▯value▯UPPER→);print(mu,4,5);
                newline(2);writetext(←critical▯value▯LOWER→);print(mu,4,5);

        newline(1);
goto init;
end of parametrize;


ra:=34359738368;re:=13743895347?;rb:=68719476736;rd:=13543228579;re:=1/ra;

rop:
select input(1); select output(1);
paperthrow;tot:=instr/2048;no:=read;writetext(←prob▯no→);print(no,4,0);
newline(2);
 m:=read; n:=read; t:= read;
```

```
        begin integer array Pt,vcol,vrow[0:m+n];
                real array str[0:m+n], a[0:(m+n+1)*(m+n+2)/2];
        real an,Pr;
        boolean  parametric;
        integer i,j,k,l;input(a,vcol,vrow,m,n,Pt);
        for i:=0 step 1 until m+n do str[i]:=a[Pt[i]+0];

initial:itor:=0;Tst:=true;ch1:=vcol[0];ch2:=vrow[0];
        dual:=primal:=false; if t=0 then Phno:=2;

Setparametor:   if itor>0 and t>0 then begin t:= t-1 ;m:=m-1 end;
        if t>0 then goto rule1;
                mk:= true;
        if not dual then
                for i:=1 step 1 until n+m do
                ifvrow[i]=vrow[0]and a[Pt[i]+0]< 0  then mk:= false;
                dual:=mk;
        if dual then
                begin mk:= true;fori:= 1 step 1 until n+m do
                if vrow[i]=vcol[0] anda[Pt[i]+0]<0 then mk:= false;
                primal:=mk;

        if primal then goto  soln;comment The above is solution exit;

ifch1≠vrow[0] then begin ch1:=vrow[0];ch2:=vcol[0] end;
        end;

if t>0 then  Phno:=1 else Phno:=if dual then2 else 3;

comment the following produces the pivot rules;
        if Phno>1 then goto rule2;
        rule1:  outgoing:=m+n;j:=random*(m+n);if j<1 then j:=1;
        for incoming:=j step -1 until 1 do
        ifvcol[incoming]=vcol[0]anda[Pt[outgoing]+incoming]≠0 then goto double;
        forincoming:=j step 1 until n+m do
        if vcol[incoming]=vcol[0] and
        a[Pt[outgoing]+incoming]≠0 then goto double;
        error: writetext(←pivot rule has gone wrong→);writetext(←j=→);
        print(j,5,0); newline(1); goto soln;

                rule2:incoming:= ifnot Tst then incoming else largest(ch2,n,Pt,n,
                                                                m,vrow);
        outgoing:=outgo(a,Pt,ch2,vrow,vcol,incoming,m,n,unbound,infeas);
```

```
if incoming=outgoing then
        begin
                Tst:=true;compi:=incoming; gotosingle
        end;
        ifabs(a[Pt[outgoing]+outgoing])>0.000000001 then
        begin
                Tst:=false;compi:=outgoing;goto single
        end;
        double:         writetext(←double$iteration→);


        Tst:= true;monitor(a,iter,0,m+n,vcol,vrow,Pt);
                duopivot(a,incoming,outgoing,m+n,Pt,iter,vcol,vrow);
        goto Setparameter;
        single:                 monitor(a,iter,compi,m+n,vcol,vrow,Pt);
                        pivot(a,compi,m+n,Pt,iter,vcol,vrow);
        goto Setparameter;

infeas:         writetext(←solution$infeasible→); goto soln;
unbound:        writetext(←solution$unbounded→);
soln:   primal:=dual:=true;
        monitor(a,iter,0,m+n,vcol,vrow,Pt);

newline(1); write text(←last$tableau$was$optimal→);newline(2);
        writetext(←instruction$count→);print(instr/2048-tot,5,0);newline(2);
        select input(2);
        parametric:=readBoolean;
        if parametric then
        begin

                i:=read;Pr:=read;
                if i< n then Pr:=-Pr;an:=str[i]; str[i]:=Pr;

        paperthrow;newline(1);writetext(←rowno=→);print(i,4,0);
        writetext(←parameter value→);print(Pr,5,5); newline(1);
                parametrize(a,m,n,Pt,vcol,vrow,i,Pr,inital,an);
        select input(1);
        end;
        end of blok;

        if no>1 then \\goto rep;
        end of program;
***Z
```

APPENDIX 10

```
JOB
LRR91PA1,SYM-QPROG TEST B/B MITRA 14.2.68.
OUTPUT
0 LINEPRINTER 450 LINES
1 LINEPRINTER 500 LINES
INPUT
0 SYM-QPROG
1 SYM-DATA
STORE 44/99 BLOCKS
COMPUTING 15500 INSTRUCTIONS
***C
COMPILER ALGOL
SYM-QPROG



begin real ro;
        real toler,temp;
        boolean primal,dual,Tst,mk,intst,bkmk,opt;
        integer m,n,no,tot,indx,hfindx,t,iter,Phno,ch1,ch2,outgoing,
        incoming,compi,ra,rb,rc,rd,int,longth,lim,pnt;
        real array store[1:4096],b[0:80];



real procedure instr;
        begin code
        1136,0,0,0;
        356,0,-,instr;
        end of instr;

real procedure random;
        begin
        rd:=5*rd; if rd>rc then rd:=rd-rc;
                if rd>rb then rd:=rd-rb;
                if rd>ra then rd:=rd-ra;
                random:=ro*rd;
end random;



integerprocedurereadch;
begin code
 0121,79,127,1;
 0101,127,0,2;
 0124,80,80,0;
 1524,0,80,0;
 0320,0,0,*14000070;
 0356,0,-,readch;
endreadch;
```

```
procedureprintch(i);
valuei;integeri;
begin codo
 1300,80,-,i;
 0172,80,0,130;
 0121,79,0,0;
 0226,79,0,*4;
 0163,80,79,0;
 0121,79,127,1;
 0101,127,0,0.4;
 0121,0,0,0;
endprintch;

procedure send(a,store,length,Pt,vcol,vrow);
          integer length, Pt;
          integer array vcol,vrow;
          real array  store,a;
        begin integer i,j,k,l; j:=length-indx-1;
        k:=length-1;l:=length-hfindx-1;
                for i:=0 step 1 until j do
                                store[Pt+i]:=a[i]
;
        fori:=j+1 step 1 until l do
                store[Pt+i]:=vcol[i-j-1];
        fori:=l+1step 1 until k do
                store[Pt+i]:=vrow[i-l-1];
        end of send;


procedure fetch(a,store,length,Pt,vcol,vrow);
           integer Pt,length;
           integer array  vcol,vrow;
           real array store,a;
        begin integer i,j,k,l;j:=length-indx-1;k:=length-1;
                        l:=length-hfindx-1;

                        for i:=0 step 1 until j do
                                a[i]:=store[Pt+i];
        fori:=j+1 step 1 until l do vcol[i-j-1]:=store[Pt+i];
        fori:=l+1 step 1 until k do vrow[i-l-1]:=store[Pt+i];
        end of fetch;


procedure add(avlist,pnt);value pnt;
           integer pnt;
           integer array avlist;

        begin integer k;
        avlist[0]:=avlist[0]+1;
        if pnt≠avlist[0] then
        begin
        k:=avlist[avlist[0]];
```

```
        avlist[avlist[0]]:=avlist[pnt];
        avlist[pnt]:=k
        end
        end of procedure;



procedure enlargetabl(a,Pt,vcol,vrow);

        real array a;
        integer array Pt,vcol,vrow;
                begin integer   i,j,k;
        for i:= n+m+1  step 1 until n+m+int do

        begin
                k:=i-n-m;
                ifvrow[k]=vrow[0] then
                    for j:= 1step 1 until  i do
                a[Pt[i]+j]:= if j=k then 1 else 0;

                ifvrow[k]≠vrow[0]then
        begin
                for j:= 1 step 1 until k do
                   a[Pt[i]+j]:=-a[Pt[k]+j];
                forj:= k+1 step 1 until  i do
        a[Pt[i]+j]:=ifvrow[j]=vcol[k] then a[Pt[j]+k]else-a[Pt[j]+k];
        end;
        a[Pt[i]+0]:=999;
        end;

                m:=m+int
        end of enlargetabl;
procedure monitor(a,it,k,limit,vcol,vrow,Pt);value it,k,limit;
        real array a;
        integer array vrow,vcol,Pt;
        integer it,k,limit;

        begin integer i,j;

                newline(2);
                writetext(←iter∕no=→);print(it,4,0);newline(2);space(11);
        for i:=0 step 1 until limit do
        begin  printch(vcol[i]);print(i,2,0);space(10);
        end;
                for i:=0 step 1 until limit do
        begin
                newline(1);printch(vrow[i]);print(i,2,0);
                for j:=0 step 1 until i do print(a[Pt[i]+j],4,8);
                if i=k then   writetext(←<--→);
        end;
out:    end of monitor;
```

```
procedure input(a,vcol,vrow,m,n,Pt,int)\\\\;value m,n,int;
        integer m,int,n;real array a;
        integer array vcol,vrow,Pt;
        begin integer i,j;
comment procedure input \\reads and sets up the starting state and
        tableau.The next bit sets up the one dimensional symmetric matrix;
        Pt[0]:=0;
        for i:=1 step 1 until n+m+int doPt[i]:=Pt[i-1]+i;
comment the objective and the p-vector follows;
        a[Pt[0]+0]:=0;
        for i:=1 step 1  until n do
        begin a[Pt[i]+0]:=read;a\\[Pt[i]+0]:=a[Pt[i]+0]*(-1)
        end;
        for i:=1 step 1 until n do
        for j:=1 step 1 until i do
                                a[Pt[i]+j]:=read;
comment the above assumes C isprocented with a negative sign;
        for i:= n+1 step 1 until n+m do
        for j:=0 step 1 until i do
                if j< n thena[Pt[i]+j]:=read else a[Pt[i]+j]:=0;

        for vcol[\\0]:=readchwhile vcol[0]=1 or vcol[0]=128 do;
        forvrow[0]:=readch while vrow[0]=1 or vrow[0]=128 do;

        for i:=0 step 1 \\ until n+m+int do
        begin if i<n then
                        begin
                        vrow[i]:=vrow[0];vcol[i]:=vcol[0]
                        end
                else
                        beginvcol[i]:=vrow[0];vrow[i]:=vcol[0]
                        end
        end;
end of input;


procedure\\ pivot(a,k,limit,Pt,iter,vcol,\\vrow);value limit,k;
        real array a;
        integer array vcol,vrow,Pt;
        integer  limit,iter,k;
        begin integer i,j,sgn; real Div;
        Div:=a[Pt[k]+k];
        for i:=0 step 1 until k-1 do
        beginsgn:=ifvcol[k]≠vrow[i] then 1 else -1 ;
        for j:= 0 step 1  until i do
                a[Pt[i]+j]:=a[Pt[i]+j]-sgn*(a[Pt[k]+j]*a[Pt[k]+i])/Div
        end;
```

```
        fori:=k+1 step 1  until limit do
        for j:=i step -1  until 0 do
        begin if j>k then begin sgn:=if vcol[j]≠vrow[k]then 1 else -1;
                a[Pt[i]+j]:=a[Pt[i]+j]-(sgn*a[Pt[j]+k]*a[Pt[i]+k])/Div;
                                end;
                if j<k thena[Pt[i]+j]:=a[Pt[i]+j]-(a[Pt[k]+j]*a[Pt[i]+k])/Div
        end;

        for i:=0 step 1 until limit do
                begin
        if i<k thena[Pt[k]+i]:=a[Pt[k]+i]/Div;
        if i>k thena[Pt[i]+k]:=-a[Pt[i]+k]/Div;
                end;
        a[Pt[k]+k]:=1/a[Pt[k]+k];

        sgn:=vrow[k];vrow[k]:=vcol[k];vcol[k]:=sgn;
        iter:=iter+1;
        end of pivot;

integer procedure largest(ch2,a,Pt,n,m,vrow);value n,m,ch2;
        integer ch2,n,m;
        integer array Pt,vrow;real array a;
        begin integer i,j; real stay;

        stay:=a[Pt[0]+0];a[Pt[0]+0]:=0;j:=0;
        for i:= 1 step 1 until  n+m do
        ifvrow[i]=ch2 and a[Pt[i]+0]< a[Pt[j]+0] then
        begin  j:=i;
        if a[Pt[j]+0]<0then goto nxt
        end;
nxt:            largest:=j;a[Pt[0]+0]:= stay
        end of largest;

integer procedure outgo(a,Pt,ch2,vrow,vcol,in,m,n,unbound,infeas);value ch2,in,m
                integer array vrow,vcol,Pt;
                real array a;
                integer in,m,n,ch2;

        label unbound,infeas;
begin real ratio,nxt,comp;
        integer i,sgn,j;
        integer arraylist[0:m+n];
        ratio:=1000000000;nxt:=2000000000;

        ifa[Pt[in]+in]<-0.00000000001 and a[Pt[in]+0]<-0.00000000001
        then begin ratio:=a[Pt[in]+0]/a[Pt[in]+in];j:=in
                end;
        for i:= 1 step 1 until in do
```

```
        ifa\\\\\bs(a[Pt[in]+i])>0.0000000001 and vro\\w[i]≠ch2 then
begin sgn:=if vcol[i]=vrow[in]then -1 else 1;
        if a[Pt[i]+0]>-\\0.000000001 and sgn*a[Pt[in]+i]<0 then goto skips;
        comp:=sgn*a[Pt[i]+0]/a[Pt\\\\[in]+i];
        if comp>-0.000000001 and comp- ratio<0.000000001 then
        begin
        if ratio-comp<0.000000001 thennxt:= ratio;
         j:=i;ratio:=comp
        end;
skips:  end;


        for i:=in step 1 until n+m do
        ifa[Pt[i]+in]≠0 and vrow[i]≠ch2 then
begin
        if a[Pt[i]+0]>-0.000000001 and a[Pt[i]+in]<0 then goto skip;
        comp:=a[Pt[i]+0]/a[Pt[i]+in];
        ifcomp>-0.000000001 and comp-ratio<0.000000001 then
        begin
        if ratio-comp<0.000000001thennxt:= ratio;
        j:=i; ratio:=comp;
        end;
skip:       end;
        outgo:=j;


        if nxt-ratio<0.00000001 then  begin
        list[0]:=0;
        for i:= 1 step 1 until in do
                if a[Pt[in]+i]≠0  and vrow[i]≠ch2 then

        beginsgn:=ifvcol[i]=vrow[in]then-1 else 1;
        ifa[Pt[i]+0]>-0.000000001 and sgn*a[Pt[in]+i]<0 then goto kips;
                comp:=sgn*a[Pt[i]+0]/a[Pt[in]+i];
        if comp>-0.000000001 andcomp-ratio<0.000000001then

                beginlist[0]:=list[0]+1;
                        list[list[0]]:=i;
                end;
kips:   end;

        fori:=m+n step-1 until in+1 do
                ifa[Pt[i]+in]≠0and vrow[i]≠ch2
        then
                begin
                ifa[Pt[i]+0]>-0.000000001 anda[Pt[i]+in]<0 then goto kip;
        comp:=a[Pt[i]+0]/a[Pt[i]+in];
        ifcomp>-0.000000001and comp-ratio<0.000000001 then
        begin list[0]:=list[0]+1;
                list[list[0]]:=i;
        end;
kip:    end;

        j:=list[0]*random; ifj=0 thenj:=1;
        outgo:=list[j];
end;
```

```
        if ratio>10000000 then begin
                if ch2=vrow[0] then goto unboundelse goto infeas
                                end;
end of procoduro;
procoduro duopivot(a,in,out,lmt,Pt,iter,vcol,vrow);value in,out,lmt;
        intogor in,out,lmt,iter;integer array vcol,vrow,Pt;
        real array a;
        bogin real array rhi,rlo,chi,clo,srhi,srlo,schi,sclo[0:lmt];
        intogor i,j,lo,hi;
        lo:=in;hi:=out;if lo>hi then beginlo:=hi;hi:=in end;
        for i:=0 step 1 until lmt do
begin
        if i<lo then beginrlo[i]:=a[Pt[lo]+i];rhi[i]:=a[Pt[hi]+i];
                clo[i]:=ifvrow[i]=vcol[lo]then-a[Pt[lo]+i]elsea[Pt[lo]\\+i];
                chi[i]:=ifvrow[i]=vcol[hi]then-a[Pt[hi]+i]elsea[Pt[hi]+i];
                        ond;
        if i>lo andi< hi then begin
                rlo[i]:=ifvcol[lo]=vrow[i]then \\\\-a[Pt[i]+lo]elsea[Pt[i]+lo];
                rhi[i]:=a[Pt[hi]+i];clo[i]:=a[Pt[i]+lo];
                chi[i]:=ifvcol[hi]=vrow[i]then-a[Pt[hi]+i]\\elsea[Pt[hi]+i];
                                end;

        if i> hi then begin rlo[i]:=ifvcol[lo]=vrow[i]then\\-a[Pt[i]+lo]
                                        elsea[Pt[i]+lo];
                rhi[i]:=ifvcol[hi]=vrow[i]then-a[Pt[i]+hi] else a[Pt[i]+hi];
                clo[i]:=a[Pt[i]+lo];
                chi[i]:=a[Pt[i]+hi]\\;
                        ond;
end;

for i:=0 step 1 until lmt do
beginsrlo[i]:=rlo[i]-rhi[i]*clo[lo]/rhi[lo];
        srhi[i]:=rhi[i]/rhi[lo];
        sclo[i]:=-clo[i]/clo[hi];
        schi[i]:=chi[i]-clo[i]*rhi[hi]/rhi[lo];
end;
        srlo[lo]:=sclo[lo];schi[hi]:=srhi[hi];
        srhi[lo]:=sclo[hi]:=1/rhi[lo];


for j:=0 step 1 until lmt do
        if j≠hi or j≠ lo then
        for i:= j step 1 until lmt do if i≠hi or i≠lo
        then
        a[Pt[i]+j]:=a[Pt[i]+j]-rhi[j]*clo[i]/rhi[lo]-srlo[j]*schi[i]/schi[lo];

        for i:= 0  step 1 until lmt do

begin if i≠ hi then
        rhi[i]:=srhi[i]-srlo[i]*schi[hi]/schi[lo] else
        rhi[i]:=-srhi[i]/schi[lo];

        if i< lo then a[Pt[lo]+i]:=rhi[i] else
        a[Pt[i]+lo]:=if vcol[i]=vrow[lo] then rhi[i] else-rhi[i];

        ifi≠hi then
        rlo[i]:=srlo[i]/srlo[hi]else rlo[i]:=1/srlo[hi];
```

```
        if  i< hi then a[Pt[hi]+i]:=rlo[i] else
        a[Pt[i]+hi]:=if vcol[i]=vrow[hi] then  rlo[i] else-rlo[i];
end;
        a[Pt[lo]+lo]:=rhi[hi];a[Pt[hi]+hi]:=rlo[lo];a[Pt[hi]+lo]:=rlo[hi];
i:=vcol[lo];vcol[lo]:=vrow[lo];vrow[lo]:=i;
i:=vcol[hi];vcol[hi]:=vrow[hi];vrow[hi]:=i;
iter:=iter+1;
end of proc duopivot;


ra:=34359738368;rc:=137438953472;rb:=68719476736;rd:=13543288579;re:=ra;rc:=1/ra;

rep:

select input(1); select output(1);
paperthrow;tot:=instr/2048;no:=read;writetext(←prob#no→);print(no,4,0);
newline(2);
 m:=read; n:=read; t:= read;int:=read;toler:=read;
indx:=2*(m+n+int+1);hfindx:=m+n+int+1;
length:=(m+n+int+1)*(m+n+int+2)/2+int+indx;lim:=entier(4096/length);
        begin integer array Pt,vcol,vrow,idn,bvrow,bvcol[0:m+n+int];
                real array a[0:(m+n+int+1)*(m+n+int+2)/2+int],cbost[0:n+m+int
                integer array avlist[0:lim];
        integer i,j,k,l;input(a,vcol,vrow,m,n,Pt,int);

iter:=0;intst:=false;Tst:=true;ch1:=vcol[0];ch2:=vrow[0];
        dual:=primal:=false; if t=0 then Phno:=2;



Setparamoter:   if iter>0 and t>0 then begin t:= t-1 ;m:=m-1 end;
                mk:= true;
        if not dual then
                for i:=1 step 1 until n+m do
                ifvrow[i]=vrow[0]and a[Pt[i]+0]< 0  then mk:= false;
                dual:=mk; goto conk;
dul:

goto conk;
if intst then begin
                newline(1);writetext(←current#state#of#parameters→);
        newline(1);writetext(←lim##length##m→);
        newline(1);print(lim,4,0);print(length,4,0);print(m,4,0);
newline(2);writetext(←the#stack→);newline(1);print(avlist[0],4,0);
newline(1);writetext(←last#pointers→);
        for i:=lim step -1 until avlist[0] doprint(avlist[i],4,0);
```

```
newline(1);writetotext(⟨backingstore⟩);writeBooloan(bkmk);
newline(1);writetotext(⟨problems⎵in⎵the⎵list⟩);nowline(1);
fori:=lim step -1 until avlist[0]+1 do
begin
        fetch(b,storo,length,avlist[i],bvcol,bvrow);
        monitor(b,itor,0,m+n,bvcol,bvrow,Pt);
        newline(1);writetext(⟨lowerlimit⟩);
        for j:=1 step 1 until int doprint(b[length-indx-int-1+j],4,0);newline(

end
end;

conk:
if dual then
                begin mk:= true;fori:= 1 step 1 until n+m do
                if vrow[i]=vcol[0] anda[Pt[i]+0]<0 then mk:= false;
                primal:=mk;

        if primal then goto  soln;comment The above is solution exit;


ifch1≠vrow[0] then begin ch1:=vrow[0];ch2:=vcol[0] end;
        end;


if t>0 then  Phno:=1 else Phno:=if dual then2 else 3;


comment the following produces the pivot rules;
        if Phno>1 then goto rule2;
        rule1:  outgoing:=m+n;j:=random*(m+n);ifj<1 then j:=1;
        for incoming:=j step -1 until 1 do
        ifvcol[incoming]=vcol[0]anda[Pt[outgoing]+incoming]≠0 then goto double;
        forincoming:=j step 1 until n+m do
        if vcol[incoming]=vcol[0] and
        a[Pt[outgoing]+incoming]≠0 then goto double;
        error: writetext(⟨pivot⎵rule⎵has⎵gone⎵wrong⟩);writetext(⟨j=⟩);
        print(j,5,0); newline(1); goto soln;

        rule2:incoming:= ifnot Tst then incoming else largest(ch2,a,Pt,n,m,vrow);
        outgoing:=outgo(a,Pt,ch2,vrow,vcol,incoming,m,n,unbound,infeas);



if incoming=outgoing then
        begin
                Tst:=true;compi:=incoming; gotosingle
        end;
        ifabs(a[Pt[outgoing]+outgoing])>0.0000000001 then
        begin
                Tst:=false;compi:=outgoing;goto single
        end;
```

```
        double:         writetext(←double⌀iteration→);
        Tst:= true;monitor(a,iter,0,m+n,vcol,vrow,Pt);
                duopivot(a,incoming,outgoing,m+n,Pt,iter,vcol,vrow);
        goto Setparameter;
        single:         monitor(a,iter,compi,m+n,vcol,vrow,Pt);
                        pivot(a,compi,m+n,Pt,iter,vcol,vrow);
        goto Setparameter;


comment the coding for branch and bound starts here;
infeas:         if not intst then
                beginwritetext(←problem⌀initially⌀infeasible→);goto exit
                end;  writetext(←infeasible→);

                goto starch;
unbound:        writetext(←problem⌀unbounded→);goto exit;
soln:   monitor(a,iter,0,m+n,vcol,vrow,Pt);
        newline(1);writetext(←lower⌀limit→);
        for i:= 1 step 1 until int\\ do print(a[length-int-indx-1+i],4,0);
        if not intst then goto integral;
        goto test;
integral:       writetext(←the⌀continuous⌀optimum→);

        intst:=true;
        if  int=0 then goto exit; avlist[0]:=lim\\-1;
        for i:= 1 step  1 until avlist[0]+1 do
        avlist[i]:=(i-1)*length;
        for i:= 1 step 1 until  int do
        a[length-indx-int-1+i]:=0;
        bkmk:= false;opt:=false;
        cbest[0]:=-99999999;
        enlargetabl(a,Pt,vcol,vrow);
        monitor(a,iter,0,m+n,vcol,vrow,Pt);
        goto dichotomy;

test:   for i:=1 step 1 until int do
        ifa[Pt[i]+0]-entier(toler/10\\+a[Pt[i]+0])>tolerand vrow[i]≠vrow[0]
        then goto   transfer;
        writetext(←a⌀ feasible⌀ solution→);
        monitor(a,iter,0,m+n,vcol,vrow,Pt);
        opt:= true;
        if a[0]>cbest[0] then begin
        for i:=1 step 1 until  n+m do
                begin
                ifi>int \\\\\\then
                  begin
                cbest[i]:\\=if vrow[i]=vrow[0] then 0 elsea[Pt[i]+0];
```

```
                    end
                \\\\\else
                begin
                    cbost[i]:=ifvrow[i]=vrow[0] thena[length-indx-1-int+i]else
                        a[Pt[i]+0]+a[length-indx-1-int+i]
                end
                end;

        cbost[0]:=a[0]
end;
starch:         ifavlist[0]<lim-1 then goto flip;
                if bkmk then goto flip;
out:    if not opt then begin
                        writetext(⟨no⎵feasible⎵integer⎵solution⟩);
                        goto exit
                        end;
                writetext(⟨optimum⎵solution⟩);goto exit;



transfer:       pnt:=avlist[avlist[0]];
                send(a,store,length,pnt,vcol,vrow);
                avlist[0]:=avlist[0]-1;
flip:           ifbkmk then goto flop; pnt:=lim-1;
        fori:= lim-1 step -1 until avlist[0]+1 do
                ifstore[avlist[i]]>store[avlist[pnt]] then pnt:=i;
        ifstore[avlist[pnt]]<cbest[0] then goto out;

        fetch(a,store,length,avlist[pnt],vcol,vrow);
        add(avlist,pnt);

dichotomy:      j:=int*random;if j=0 then j:=1;if j\\=2then j:=1;
        for i:= j step -1 until 1 do
                if vrow[i]≠vrow[0]anda[Pt[i]+0]-entier(toler/10+a[Pt[i]+0])>toler
        then goto \\nxt;
        for i:= j step 1 until int do
                ifvrow[i]≠vrow[0]anda[Pt[i]+0]-entier(toler/11\\+a[Pt[i]+0])>toler
        then gotonxt;

errors:nowline(1);writetext(⟨no⎵dichotomy⎵possible⟩);goto exit;
nxt:    j:=a[length-indx-int-1+i];
        a[length-int-indx-1+i]:=entier(a[Pt[i]+0])+1+a[length-int-indx-1+i];
        temp:=a[Pt[i]+0];
        a[Pt[i]+0]:=-1-entier(temp)+temp;
        send(a,store,length,avlist[lim],vcol,vrow);
        bkmk:=true;

        a[Pt[i]+0]\\:=temp;
        a[Pt[n+m-int+i]+0]:=-temp+entier(temp);
        a[length-indx-int-1+i]:=j;
                goto dul;
flop:   fetch(a,store,length,avlist[lim],vcol,vrow);
        bkmk:=false ;
        goto dul;
```

```
exit:   monitor(a,iter,0,m+n,vcol,vrow,Pt);
        newline(2);writetext(←objective=→);print(cbest[0],5,4);n\\ewline(2);
        for i:=1 step 1 until  ndo
                begin
                        newline(1);printch(vcol[0]);print(i,3,0);
                        \\ print(cbest[i],4,4);
                end;

        writetext(←instruction↓count→);print(tot-\\\\instr/2048,5,0);
                end of blok;

        if no>1 then goto rep;
end of program;

***Z
\\\\\\\\\\
```

APPENDIX 11

```
JOB
LRR91PA1, MITRA/ F.C. TRANSPORT/DICHOTOMIES. MITRA 9.10.67
INPUT
0 TRANSPORT
1 TRSP/TABLEAU
OUTPUT
0 LINEPRINTER 500 LINES
1 LINEPRINTER 800 LINES
STORE 50/95 BLOCKS
COMPUTING 6000 INSTRUCTIONS
***C

COMPILER ALGOL
TRANSPORT

begin integer m,n,nfin, length, lamstr, factor, lim;
        reallam;
         boolean bkmk;
                integer array store[1:15000];

        procedure input(fc,a,b,m,n); value m,n;
        integer m,n;
        \\\\\\\\\\\\\\integer array fc,a,b;
        begin integer i,j;
        for i:=1 step 1 until m do
beginfor j:=1 step 1 until n do
        begin
fc[i,j,0]:= read; fc[i,j,1]:= read
end;
a[i]:= read;
end;
for j:=1 step 1 until n do b[j]:= read;
end procedure input;

procedure scale(fc,c,\\factor,a,b);
        integer factor;
        integer array fc,c,a,b;
        begin real array crel[1:m,1:n];
                integer i,j,k;
                boolean mk;
                real eps;
        for i:=1 step 1 \\until m do
        for j:=1 step 1 until n do
begink:= if a[i]<b[j] then a[i] else b[j];
        crel[i,j]:=  fc[i,j,0]+ fc[i,j,1]/k;
end;
        factor:=1;
rep: for factor:= factor*10 whilefactor ≠ 100000 and not mk do

begin           mk:= true;
        for i:=1 step 1 until m do
        for j:=1 step 1 until n do
   begin eps:= crel[i,j]*factor- entier(crel[i,j]*factor);
        if eps> 0.000001 and eps< 0.999999 then begin
                mk:= false;goto rep
                                                end;
        end;
end;
```

```
        factor:= factor\\\\/10;
for i:=1 step 1 until m do
for j:=1 step 1 until n do
c[i,j]:= crel [i,j]*factor;
end of scale;

procedure monitorx(x,kw,factor,a,b); value kw, factor;
                integer kw,factor;
                integer array x,a,b;
   begin integer i,j;
         newline(1);
     writetext(←x␣tableau␣and␣a␣b␣array→); newline(1);
     for i:= 1 step 1 until m do
begin newline(1);
     for j:= 1 step 1 until n do
                                print(x[i,j],4,0);
                writetext(←*→); print(a[i],4,0);
end;
     newline(1); for j:=1 step 1 until 9*n do writetext (←*→);
     newline(1);
     for j:= 1 step 1 until n do print(b[j],5,0);
     newline(1);
   writetext(←lower␣bound␣lambda=→);
    print(kw/factor,5,6);
end of monitorx;

procedure monitorprob(c, factor,s,a,b); value factor,s;
                integer factor,s;
                integerarray c,a,b;
begin \\\\\\\\\\\\integer i,j;

                newline(1);
                writetext(←problem␣matrix→); space(6);writetext(←set up ␣cost=→);
                print(s,4,0);
                newline(1);
for i:=1 step 1 until m do
   begin newline(1);
      for j:=1 step 1 until n do
                print(c[i,j],11,0);
        writetext (←*→);print(a[i],4,0);
   end;
     newline(1);
  for i:=1 step 1 until 15*n do writetext(←*→);
  newline(1);
  for j:=1 step 1 until n do print(b[j],4,0); space(6);
  writetext(←factor=→);
  print(factor,5,0);
end monitorprob;
```

```
procedure send(c,store,s,imp,pt); value s,pt;
                integer s,pt;
              integer array c,store,imp;
begin integer  i,j,k,l;

        for i:=1 step 1 until m do
          for j:=1 step 1 until n do
   begin
        k:= (i-1)*n+j-1;
     store[pt+k]:=c[i,j];


   end;

   store[pt+k+1]:=s;
   k:=k+2;
   for i:= 0 step 1 until n+m do

for j:= 0,1 do
begin
        l:=i*2+j;
        store[pt+k+1]:= imp[i,j];

end;

end of proc send;



procedure fetch(c,store,s,imp,pt); value pt;
        integer pt,s;
        integer array c, store, imp;
   begin integer i,j,k,l;


     for i:=1 step 1 until m do
     for j:=1 step 1 until n do
begin
        k:=(i-1)*n+j-1;
        c[i,j]:= store[pt+k];

   end;
 s:=store[pt+k+1];


    k:=k+2;
   for i:=0 step 1 until n+m do

     for j:=0, 1 do
  begin
        l:=i*2+j;
        imp[i,j]:= store[pt+k+1];

  end;

end of procedure fetch;
```

```
procedure transpi(m,n,inf,c,a,b,x,kw,factor); value a,b,c,m,n,inf,factor;
        integer m,n,inf,kw,factor; integer array c,a,b,x;
comment comm A C M vol9/NO12/DEC1966/p869;
begin integer i,j,u,v,k,l,s,t,gd,h,p,cij,xij,ai,bj,lsvj,nlvi;
        boolean ng;
        integer array g,listu,nlv[1:m],r,listv[1:n],ls[0:m+n-1],
                nl[1:m*n],lsv[0:n];

        procedure in;
                begin
                lsvj:=lsv[j];
        for t:=lsv[n] step -1 until lsvj do ls[t+1]:=ls[t];
        fort:=j step 1 until n do lsv[t]:=lsv[t]+1;
                ls[lsvj+1]:=i;
        end;

procedure out;
begin
        lsvj:=lsv[j];
        for t:=lsv[j-1]+1 step 1 until lsvj do
        begin
                if ls[t]≠i then goto next;
                s:=t; goto ex;
        next:
        end;
ex: for t:=j step 1 until n do lsv[t]:=lsv[t]-1;
        lsvj:=lsv[n];
        for t:=s step 1 until lsvj do ls[t]:=ls[t+1]
        end;


comment procedure body begins;
        for i:=1 step 1 until m do
        for j:=1 step 1 until n do x[i,j]:=0;
for i:=1 step 1 until m do nlv[i]:=(i-1)*n;
lsv[0]:=0;
for j:=1 step 1 until n do
begin
        listv[j]:=1;
        lsv[j]:=0;
end;

s1: kw:=gd:=0;
        for i:=1 step 1 until m do
        begin
                h:=inf;
                for j:=1 step 1 until n do
                if c[i,j]<h then h:=c[i,j];
                for j:=1 step 1 until n do
```

```
begin
        cij:=c[i,j]:=c[i,j]-h;
        if cij=0 then
begin
        listv[j]:=0;
        nlvi:=nlv[i]:=nlv[i]+1;
        nl[nlvi]:=j
end
end;
kw:=h*a[i]+kw
end;
for j:= 1 step 1 until n do
begin
        if listv[j]=0 then goto nextj1;
        h:=inf;
for i:=1 step 1 until m do
        if c[i,j]<h then h:=c[i,j];
for i:= 1 step 1 until m do
begin
        cij:=c[i,j]:=c[i,j]-h;
if cij=0 then
begin
        nlvi:=nlv[i]:=nlv[i]+1;
        nl[nlvi]:=j
end
end;
kw:=h*b[j]+kw;
next j1:
end;


s2: for i:=1 step 1 until m do
        begin
                ai:=a[i];nlvi:=nlv[i];
        for u:=(i-1)*n+1 step 1 until nlvi do
begin
        if ai=0 then goto next i2;
                j:=nl[u];
bj:=b[j];
if bj=0 then goto next j4;
h:=x[i,j]:= if ai<bj then ai else bj;
        ai:=ai-h;b[j]:=bj-h;in;
next j4:
end;
next i2:a[i]:=ai; gd:=gd+ai
end;

s31:
        if gd=0 then goto s6;
s32:
        for j:=1 step 1 until n do r[j]:=0;
        k:=0;
        for i:=1 step 1 until m do
```

```
begin
        if a[i]≠0 then
        begin
                k:=k+1;listu[k]:=i;g[i]:=inf
end
        else g[i]:=0;
end;

s33:l:=0;
        for u:=1 step 1 until k do
        begin
                i:=listu[u]; nlvi:=nlv[i];
for s:=(i-1)*n +1 step 1 until nlvi do
        begin j:=nl[s];
                if r[j]≠0 then goto next j5;
                        r[j]:=i;l:=l+1;listv[l]:=j;
                if b[j]>0 then goto s4;
        next j5:
        end;
        end;
if l=0 then goto s5;
        k:=0;
        for v:=1 step 1 until l do
        begin
                j:=listv[v]; lsvj:=lsv[j];
                for s:=lsv[j-1]+1 step 1 until lsvj do
        begin
                i:=ls[s];
                if g[i]=0 then
                begin
                        g[i]:=j;k:=k+1; listu[k]:=i
                end
        end
end;
if k=0 then goto s5;
goto s33;

s4: h:=b[j]; p:=j;
        mark: i:=r[j]; j:=g[i];
if j=inf then
begin
        if a[i]<h then h:=a[i]; goto re
end;
if x[i,j]<h then h:=x[i,j];
goto mark;
re  :j:=p; b[j]:=b[j]-h; a[i]:=a[i]-h;
    gd:=gd-h;
re1 :i:=r[j]; xij:=x[i,j]; x[i,j]:=xij+h;
        if xij=0 then in;
j:=g[i];
        if j=inf then goto s31;
xij:=x[i,j]:=x[i,j]-h;
        if xij=0 then out;
        goto re1;
```

```
s5:     k:=0; l:=n+1;
        for j:=1 step 1 until n do
        begin if r[j]=0 then
                begin
                        k:=k+1; listv[k]:=j
                end
              else
        begin
              l:=l-1; listv[l]:=j
        end
        end;
        h:=inf;
        for i:=1 step 1 until m do
          begin
                if g[i]=0 then goto next i6;
                for s:=1 step 1 until k do
                begin
                        j:=listv[s];
                     if c[i,j]<h then h:=c[i,j]
                end;
        next i6:
                end;
        for i:=1 step 1 until m do
        begin
                zg:=g[i]≠0; nlvi:=(i-1)*n;
                for s:=1 step 1 until n do
        begin
                j:= listv[s];
           if zg then cij:=c[i,j]
             else
           cij:=c[i,j]:=c[i,j]+h;
        if cij=0 then begin
                        nlvi:=nlvi+1; nl[nlvi]:=j
                      end
        end;
        for s:=1 step 1 until k  do
        begin
        j:=listv[s];
          if zg then cij:=c[i,j]:=c[i,j]-h
            else cij:=c[i,j];
        if cij=0 then begin
                        nlvi:=nlvi+1;
                        nl[nlvi]:=j
                      end
end;
        nlv[i]:=nlvi
end;
        kw:=h*gd+kw;
goto s32;
s0   :end;
```

```
        select input (1); selectoutput (1);
more: m:=read; n:=read; nfin:=9999999999; length:= n*m+2*(n+m+1)+1;
   lim:=entier(15000/length);
begin integer array imp[0:n+m,0:1],a[1:m],b[1:n],x[1:m,1:n],
                avlist[0:lim],cbest[1:m,1:n],c[1:m,1:n],fc[1:m,1:n,0:1];
        integer kw,setup,ct best,probno,i,j,k,l;
        boolean data;
        bkmk:=false;
        ctbest:=9999999999;
        probno:=0; setup:=0;
        for i:=1 step 1 until lim do avlist[i]:= (i-1)*length;
        avlist[0]:=lim-1;
     input(fc,a,b,m,n);
     scale(fc,c,factor,a,b);
recur:\\ifprobno<\\3 thenmonitorprob(c,factor,setup,a,b);



        transp1 (m,n,nfin,c,a,b,x,kw,factor);
            lam:=kw/factor+ setup; lamstr:=0;
        for i:=1 step 1 until m do
        for j:=1 step 1 until n do
                if x[i,j]>0 then lamstr:=lamstr+fc[i,j,0]*x[i,j]+fc[i,j,1];
        probno:= probno+1;
        if probno<3 thenmonitorx(x,kw,factor,a,b);

writetext(←lam$lam*→); print(lam,5,4);print(lamstr,5,0); newline(1);
        if probno=1 then goto sets;

        if lamstr< ct best then goto transfer;
        iflam> ctbest then  goto srch;
stores: if imp[0,0]= imp[0,1] then goto srch;
        k:=avlist[avlist[0]];
send(c,\\store,setup,imp,k);
        avlist[0]:=avlist[0]-1;
srch: if bkmk then  goto flop;
   flip: if avlist[0]=lim-1then goto exit; k:=avlist[avlist[0]+1];
        fetch (c,store, setup, imp,k); avlist[0]:=avlist[0]+1;
        i:=imp[imp[0,0],0];
        j:=imp[imp[0,0],1];
newline(1);
        imp[0,0]:= imp[0,0]+1;
        c[i,j]:=factor*fc[i,j,0];
        k:=setup+fc[i,j,1];
send(c,store,k,imp,avlist[lim]);



        bkmk:=true;
        c[i,j]:=9999*factor;
          goto recur;
flop: bkmk:= false;
        fetch(c,store, setup,imp,avlist[lim]);
```

```
        goto recur;
sots:k:=0; imp[0,0]:=1;
newline(1); writetext(⟨initial▯value⟩);print(length,4,0);print(lim,4,0);
        fori:=0step 1 until avlist[0]+1 doprint(avlist[i],4,0);
        for i:=1 step 1 until m do
        for j:=1 step 1 until n do
    if x[i,j]>0 then
begin k:=k+1;
        imp[k,0]:=i;
        imp[k,1]:=j;
end;
    imp[0,1]:=k+1;
transfer:otbest:=lamstr; monitorx(x,kw,factor,a,b);

    for i:=1 step 1 until m do    for j:=1 step 1 until n do
        obest[i,j]:=x[i,j];
   goto stores;
exit:writetext(⟨optimum▯solution⟩);newline(1); print(otbest,5,0);
     for i:=1 step 1 until m do
begin newline(1);
     for j:= 1 step 1 until n do print(obest[i,j],5,0)
end;

writetext(⟨totalnoofprob⟩);print(probno,5,0);
end of block;
end of program;
***Z
```

APPENDIX 12

```
JOB
LRR91PA1,MITRA/RUNSLOAD*E*LRR91PA1/FC-MITRA
OUTPUT
0 LINEPRINTER 700 LINES
1 LINEPRINTER 100 LINES
INPUT
0 SYM-QPROG
1 SYM-FC
TAPE 99 PAS LS311*WRITE PERMIT
STORE 50/115 BLOCKS
COMPUTING 10000 INSTRUCTIONS
***C
COMPILER ALGOL
SYM-QPROG

OMIT COMPILER MONITORING;


begin real ro,setup,eps,lamup,lamlowor,clamlowor,clamup,up,low;
        boolean primal,dual,Tst,mk,mkr,empty,src,type,prnt;
        integer n,n,no,count,tot,t,iter,Phno,ch1,ch2,sino,
                outgoing,incoming,compi,ra,rb,rc,rd,max,trlength,trno,initials,
                totpr,tottrnsfr,cnt,ub,length,lim,chooseprob,choosevar,blockno;

procedure load;
begin code
113,    89,     0,      (2);
1001,   99,     0,      2;
1002,   99,     0,      *3377;
1101,   89,     0,      *3377;
*050662/2:0;
end;


real procedure instr;
        begin code
        1136,0,0,0;
        356,0,-,instr;
        end of instr;

real procedure random;
        begin
        rd:=5*rd; if rd>rc then rd:=rd-rc;
                if rd>rb then rd:=rd-rb;
                if rd>ra then rd:=rd-ra;
                random:=ro*rd;
end random;
```

```
integerprocedurereadch;
begin code
 0121,79,127,1;
 0101,127,0,2;
 0124,80,80,0;
 1524,0,80,0;
 0320,0,0,*14000070;
 0356,0,-,readch;
endreadch;

procedureprintch(i);
valuei;integeri;
begin code
 1300,80,-,i;
 0172,80,0,130;
 0121,79,0,0;
 0226,79,0,*4;
 0163,80,79,0;
 0121,79,127,1;
 0101,127,0,0.4;
 0121,0,0,0;
endprintch;

procedure searches(tape,block);
value tape,block;
integer tape,block;
comment This procedure winds 'tape' to the start of 'block';
begin code
        1300, 58, -,block;
        1300, 57, -,tape;
        0163, 57, 0,0;
        0121,121,57,0;
        1001,122,58,0;
end search;


procedure storearray(A,m,mm,n,nn,tape,block);
value   m,mm,n,nn,tape;
integer m,mm,n,nn,tape,block; array A;
comment This procedure stores the array A on magnetic tape numbered 'tape',
starting at the beginning of block numbered 'block'.
The array is taken to be A[m:mm] if nn<n and A[m:mm,n:nn] otherwise.
On exit 'block' gives the first free block of mag tape after the array;
begin
        integer size,blocklocal;
```

```
        array BUFFER[0:1023];
        procedure addr1;  begin code  B79=ADDRESS OF A[m];   0121,59,79,0 end;
        procedure addr2;  begin code  B79=ADDRESS OF A[m,n]; 0121,59,79,0 end;
        procedure store;
        begin code
                1300, 58, -,size;
                0124, 59,58,-1;
                0120, 58, 0,1;
                B79=ADDRESS OF BUFFER[512];
                0165, 57,79,-512;
                0124, 57, 0,511;
                0121, 56, 0,-511;
                1300, 55, -,tape;
                0163,55, 0,0;
                1300, 54, -,blocklocal;
                0121,121,55,0;
                1001,122,54,0;
        1:      0334, 59,58,0;
                0356, 57,56,0;
                0201,127,56,(2);
                0121,121,55,0;
                1004,122,57,-511;
                0124, 54, 0,1;
                0121,56, 0,-511;
        2:      0201,127,58,(1);
                0172, 56, 0,-511;
                0224,127, 0,(3);
                0121,121,55,0;
                1004,122,57,-511;
                0124, 54, 0,1;
        3:      1524,  0,54,0;
                0356,  0, -,blocklocal;
        end of store;

        blocklocal:=block;
        size:=(if mm>m then mm-m+1 else 1)*(if nn>n then nn-n+1 else 1);
        if nn>n then addr2 else addr1;
        store;
        block:=blocklocal;
end of storearray;
```

```
procedure fetcharray(A,m,mm,n,nn,tape,block);
value   m,mm,n,nn,tape,block;
integer m,mm,n,nn,tape,block; array A;
comment This procedure fetches the array A from magnetic tape numbered 'tape',
starting from the beginning of block numbered 'block'.
The array is taken to be A[m:mm] if nn<n and A[m:mm,n:nn] otherwise;
begin
        integer size,blocklocal;
        array BUFFER[0:1023];
        procedure addr1;  begin code  B79=ADDRESS OF A[m];   0121,59,79,0  end;
        procedure addr2;  begin code  B79=ADDRESS OF A[m,n]; 0121,59,79,0  end;
        procedure fetch;
        begin code
                1300, 58, -,size;
                0124, 59,58,-1;
                0120, 58, 0,1;
                B79=ADDRESS OF BUFFER[512];
                0165, 57,79,-512;
                0124, 57, 0,511;
                0121, 56, 0,-511;
                1300, 55, -,tape;
                0163, 55, 0,0;
                1300, 54, -,block;
                0121,121,55,0;
                1001,122,54,0;
                1002,122,57,-511;
        1:      0334, 57,56,0;
                0356, 59,58,0;
                0201,127,56,(2);
                0121,121,55,0;
                1002,122,57,-511;
                0124, 54, 0,1;
                0121, 56, 0,-511;
        2:      0201,127,58,(1);
        end of fetch;

        size:=(if mm>m then mm-m+1 else 1)*(if nn>n then nn-n+1 else 1);
        if nn>n then addr2 else addr1;
        fetch;
end of fetcharray;

integer procedure chprob(avlist,store); integer array avlist;array store;
\\\\\\\\\\\\\\\\\\\\\\\\\\
begininteger i, j;real best;best:=-999999999;
        if chooseprob=1 \\\\then goto exit;comment LIFO scheme;

if \\\\chooseprob=2 then

        begin j:=avlist[0]+entier(random*((lim-1)-avlist[0]));
                goto swop
        end;
comment random choice of problem;
if chooseprob=3  then
begin for i:=avlist[0] step 1 until lim-1 do
        ifstore[avlist[i]+length-1]>best
        then begin
                best:=store[avlist[i]+length-1];
                j:=i
```

```
                end;

    end;

swop:   i:=avlist[avlist[0]];avlist[avlist\\[0]]:=avlist[j];
        avlist[j]:=i;
exit:   chprob:=avl\\ist[0]
end of procedure;

integer procedure chvar(clist,a,Pt,vrow,fc,phis);
        integer arrayclist,Pt,vrow;
        array a,fc,phis;

        begin integer i,j;real best,rel;best:=-1;
comment random choice;
        if choosevar=1 then
        begin j:=random*n; if j=0 then j:=1;
for i:=j step -1 until 1 do
        if vrow[i]≠vrow[0] and not include(clist,i) then goto exit;


for i:=j step 1 until n do
        ifvrow[i]≠vrow[0] and not include(clist,i) then  goto exit

        end;
for j:=1 step 1 until n do
        ifvrow[j]≠vrow[0] and not include(clist,j)and a[Pt[j]+0]>0.000001

        then
        begin
           rel:=fc[j]-(a[Pt[j]+0]*fc[j])/phis[j];
        if rel> best then
                        begin
                        best:= rel;
                        i:=j
                        end;
        end;
exit:chvar:=i
end of procedure ;
boolean procedure include(clist,i);value i;
        integer array clist; integer i;

begin integer j;
include:= false ;
if clist[0]=0  then  goto exit;
for j:= 1 step 1 until clist[0] do
        if clist[j]=i then  goto jump;
goto exit;
jump: include:=  true;
exit:
end of include;
```

```
real procedure evaluate(qd,fc,a,Pt,vrow,str);
        array qd,fc,a,str;
        integer array Pt,vrow;
begin integer i; real val; val:=0;

for i := 1 step 1 until n do
        if vrow[i]≠vrow[0] and a[Pt[i]+0]>0.000001
        then
        val:=val-2*(fc[i]+str[i]*a[Pt[i]+0])+qd[i]*a[Pt[i]+0]*a[Pt[i]+0];
        evaluate:= val;
end odf evaluate;

procedure monitorbest(cbest,cchar,clist);
        array cbest;
        integer array cchar,clist;

        begin integer i;
        newline(1);writetext(←current≠best≠solution//lamup→);
        print(clamup,5,5);
        for i:= 1 step 1 until n do
beginnewline(1);printch(cchar[i]);print(i,3,0);writetext(←=→);
        print(cbest[i],5,5); writetext(←clista→);
        print(clist[i],4,0);
end
end of monitor best;
procedure monitor(a,it,k,limit,vcol,vrow,Pt,clist);value it,k,limit;
        real array a;
        integer array vrow,vcol,Pt,clist;
        integer it,k,limit;

        begin integer i,j;

        if not prnt then goto out;

newline(2);
                writetext(←iter≠no=→);print(it,4,0);newline(2);
space(11);
        if not prnt then goto out;
if limit > size then goto extra;
for i:=0 step 1 until limit do
        begin  printch(vcol[i]);print(i,2,0);space(10);
        end;
                for i:=0 step 1 until limit do
        begin
                newline(1);printch(vrow[i]);print(i,2,0);
                for j:=0 step 1 until i do print(a[Pt[i]+j],4,8);
                if i=k then   writetext(←<--→);
        end;
        extra: for i:=0 step 1 until limit do

        begin printch(vrow[i]);print(i,2,0); writetext(←=→);
                print(a[Pt[i]+0],4,5);
        end;


        ■■■
```

```
        newline\\(1);writetext(←bounds≠lamup≠lamlower→);
        print(lamup,6,6);print(lamlower,6,6);newline(1);
        writetext(←clist→);fori:=0step 1 until n do print(clist[i],3,0);newline(

        writetext(←setup→);print(setup,\\\\\\\8,4);

out:

end of monitor;


procedure clear(a,Pt);
        arraya;integer array Pt;

        begin integer i,j;
        for i:= 0 step 1 until m +2*n do
        for j:= 0 step 1 until i do
                a[Pt[i]+j]:=0;
        end of procedure clear;
\\\\\\\\\\\\\\\\
procedure sparse(a,Pt);arraya;integer array Pt;

        begin integer i,j,k,l;
        clear(a,Pt);
loop:   i:= read;if i=\\0 then goto exit; if i>m then goto error;
        j:=read;i:=i+n; if j>n then goto error;
        a[Pt[i]+j]:=read;
goto loop;

error:  newline(1);writetext(←indices≠out≠of≠bounds→);print(i,3,0);print(j,3,0);
        writetext(←IANDJ→);
exit:   end of sparse;


procedure Block(a,Pt);array a; integer array Pt;

        begin integer   i,j,k,l;
        integer array ms,ns[0:blockno];

        clear(a,Pt);ms[0]:=0;ns[0]:=0;
        if blockno=1 then

begin
        for i:= 1 step 1 until m do
        for j:= 0 step 1 until n do
                a[Pt[n+i]+j]:=read;
end

                else

begin for i:= 1 step 1 until  blockno do
```

```
                begin
                ms[i]:=read;ms[i]:=ms[i-1]+ms[i];
                ns[i]:=read;ns[i]:=ns[i-1]+ns[i];\\

        for k:=ms[i-1]+1 step 1 until ms[i] do
        for1:=0,ns[i-1]+1 step 1 until ns[i] do
                a[Pt[k+n]+1]:= read;

                end;

        if ms[blockno]≠m orns[blockno]≠n then writetext(«block error»);
end

end of procedure block;


procedure frillput(a,vcol,\\vrow,Pt,qd,fc,str,phis,avlist,cbest,cchar);

        real array      a,qd,fc,str,phis,cbest;
        integer array   vcol,vrow,Pt,avlist,cchar;
begin integer i,j,k,l;
        comment set up the list;

        for i:= 1  step  1  until  lim do
        avlist[i]:=(i-1)*length+1;
comment         read the coeffecients of the four important vectors;
        fori:= 1 step  1 until  n do str[i]:= read;
        fori:= 1 step  1 until  n do qd[i] := read;
        fori:= 1 step  1 until  n do fc[i] := read;
        fori:= 1 step  1 until  n do phis[i]:= read;


comment the following sets up the pointer;

        Pt[0]:=0;
        for i:= 1 step 1 until  m+n*2 doPt[i]:=Pt[i-1]+\\i;
type:=readBoolean; blockno:= read;
if not type thensparse(a,Pt)elseBlock(a,Pt);

comment the following sets up the tableau entries;
        for i:= 1 step 1 until  n do
begin
        a[Pt[i]+0]:=str[i]+fc[i]/phis[i];
        a[Pt[i]+i]:=qd[i];
end;

comment set up  the  bounds;
 for i := 1 step  1 until  n do
```

```
begin
        a[Pt[n+m+i]+0]:=phis[i];
        a[Pt[n+m+i]+i]:=1 ;
end;

for vcol[0]:=readch while vcol[0]=1  or vcol[0] =128 do;
for vrow[0]:=readch while vrow[0]=1  or vrow[0] =128 do;

comment  sets up the identifier markers;

        for i:= 1 step  1 until  m+2*n do
        begin if i< n  then
                begin
                   vrow[i]:=vrow[0];
                   vcol[i]:=vcol[0];
                end
        else
                begin
                  vcol[i]:=vrow[0];
                  vrow[i]:=vcol[0]
                end
        end;

type:=readBoolean;

comment if a feasible solution is at hand then input it;

if type then
        begin
                clamup:= read;
        fori:= 1 step 1 until  n do
        begin
                cbest[i]:= read;ifcbest[i]>0.000001 then
        cchar[i]:=vcol[i] else cchar[i]:= vrow[i]
        end
        end
                else clamup:=-999999999;

comment  set up the constraint nos ;
m:= n+ m;

end frillput;

procedure logic(avlist,pt,store);value pt;
integer pt;
integer array avlist;  array store;

begin integer i;

        totpr:=totpr+1;
if pt=lim then begin mkr:=true; goto exit end;
```

```
        ifmax<trno*(lim-1)/trlongth+(lim-1)-avlist[0]
        then \\\\\\\\\\\\\\\\\\max:=max+1;


        if avlist[0]≠1 then

                begin
                        if not empty then
                        avlist[0]:=avlist[0]-1
                        else empty:=false;
                        goto exit
                end;

        avlist[0]:=lim-1; empty:=false;
        for i:= 1 step 1 until lim do
        store[lim*length+i]:=avlist[i];
        i:=initials+trno;

ifnot src then begin
                writetext(←tape⁄used→);src:=true;
                searches(1,initials)
                end;
        storearray(store,1,ub,1,0,1,i);
        trno:=i-initials;
exit:   end of procedure logic;

procedure invlogic(avlist,pt,exhaust,store);value pt;
                \\\\array store;

        integer pt;
        integer array avlist;
        label   exhaust;
begin integer   i;

        ifpt=lim        then beginmkr:=false; goto exit end;

        ifavlist[0]≠lim-1 then  begin
                                avlist[0]:=avlist[0]+1;
                                goto            exit
                                end;
        if      not empty then
                                begin empty:= true; goto exit end;
        if trno=0 then goto exhaust;
        empty:= false;
        trno:=trno-trlength;

fetcharray(store,1,ub,1,0,1,initials+trno);
        fori:= 1 step 1 until lim do
        avlist[i]:=store[lim*length+i];
```

```
        avlist[0]:=2;
exit:   end of procoduro;


procedurosond(storo,a,Pt,clist,pt,vrow,avlist);valuo pt;
        intogor pt;

        intogor array Pt,avlist,clist,vrow;
        real array storo,a;
begin integer i,counter,j;
        logic(avlist,pt,storo);

        counter:=ifpt=lim then avlist[lim]elseavlist[avlist[0]];

for i:= 0 step 1 until  n+m do
        for j:=0 step 1 until i do
        store[counter+Pt[i]+j]:=a[Pt[i]+j];
        counter:=counter+(n+m+1)*(n+m+2)/2;
for i:= 0 step 1 until  n do store[counter+i]:=clist[i];
        counter:= counter +n ;
for i:= 1 step 1 until  m+n do
        storo[counter+i]:= vrow[i];
store[counter+m+n+1]:= setup;
        store[counter+m+n+2]:=lamup;
store[counter+m+n+3]:= lamlower;
end of procedure send;


procedure fetch(store,a,Pt,clist,\\\\\\\\\vrow,vcol,pt,exhaust,avlist); value pt;
        label exhaust;
        integer pt;
        integer array Pt,clist,vrow,avlist,vcol;
        real array store,a;
begin integer i,j,counter;
        invlogic(avlist,pt,exhaust,store);
        if pt=lim then counter:=avlist[lim]
                  else
        counter:=if not empty then avlist[a\\vlist[0]-1] else avlist[avlist[0]];
for i:= 0 step 1 until n+m do
for j:=0 step  1 until  i do
                a[Pt[i]+j]:=store[counter+Pt[i]+j];
                counter:=counter+(n+m+1)*(n+m+2)/2;
for i:= 0 step 1 until  n do
        clist[i]:=store[counter +i];

        counter:=counter+n;

for i:= 1 step 1 until   m+n do
```

```
begin
        vrow[i]:=store[counter+i];
        vcol[i]:=if vrow[i]=vrow[0] then vcol[0] else  vrow[0];
        end;

setup:=store[counter+n+n+1];
end of fetch;


procedure\\ pivot(a,k,limit,Pt,iter,vcol,\\vrow);value limit,k;
        real array a;
        integer array vcol,vrow,Pt;
        integer  limit,iter,k;
        begin integer i,j,sgn; real Div,rowmplr;
        Div:=1/a[Pt[k]+k];
        for i:=0 step 1 until k-1 do
        beginsgn:=ifvcol[k]≠vrow[i] then 1 else -1 ;

        rowmplr:=a[Pt[k]+i]*Div;

        for j:= 0 step 1  until i do
                a[Pt[i]+j]:=a[Pt[i]+j]-sgn*(a[Pt[k]+j]*rowmplr)
        end;

        fori:=k+1 step 1  until limit do

begin   rowmplr:=a[Pt[i]+k]*Div;
        for j:=i step -1  until 0 do
        begin

         if j>k then begin sgn:=if vcol[j]≠vrow[k]then 1 else -1;
                a[Pt[i]+j]:=a[Pt[i]+j]-sgn*a[Pt[j]+k]*rowmplr
                                end;
                if j<k thena[Pt[i]+j]:=a[Pt[i]+j]-a[Pt[k]+j]*rowmplr
        end
end;

        for i:=0 step 1 \\until limit do
                begin
        if i<k thena[Pt[k]+i]:=a[Pt[k]+i]*Div;
        if i>k thena[Pt[i]+k]:=-a[Pt[i]+k]*Div
                end;
        a[Pt[k]+k]:=1/a[Pt[k]+k];

        sgn:=vrow[k];vrow[k]:=vcol[k];vcol[k]:=sgn;
        iter:=iter+1;
        end of pivot;
```

```
integer procedure largest(ch2,a,Pt,n,m,vrow);\\\\\\\\\value n,m,ch2;
        integer ch2,n,m;
        integer array Pt,vrow;real array a;
        begin integer i,j; real stay;

        stay:=a[Pt[0]+0];a[Pt[0]+0]\\:=0;j:=0;
        for i:= 1 step 1 until  n+m do
        ifvrow[i]=ch2 and a[Pt[i]+0]< a[Pt[j]+0] then j:=i;
        largest:=j;a[Pt[0]+0]:= stay
        end of largest;


integer procedure outgo(a,Pt,ch2,vrow,vcol,in,m,n,unbound,infeas);value ch2,in,m,n
                integer array vrow,vcol,Pt;
                real array a;
                integer in,m,n,ch2;

        label unbound,infeas;
begin real ratio,nxt,comp;
        integer i,sgn,j;
        integer arraylist[0:m+n];
        ratio:=1000000000;nxt:=2000000000;

        ifa[Pt[in]+in]<-0.00000000001\\ and a[Pt[in]+0]<-0.00000000001
        then begin ratio:=a[Pt[in]+0]/a\\[Pt[in]+in];j:=in\\
                end;
        for i:= 1 step 1 until in do
        ifa[Pt[in]+i]≠0 and vro\\w[i]≠ch2 then
begin sgn:=if vcol[i]=vrow[in]then -1 else 1;
        if a[Pt[i]+0]>-0.000000001 and sgn*a[Pt[in]+i]<0 then goto skips;
        comp:=sgn*a[Pt[i]+0]/a[Pt\\\\[in]+i];
        if comp>-0.000000001 and comp- ratio<0.000000001 then
        begin
        if ratio-comp<0.000000001 thennxt:= ratio;
         j:=i;ratio:=comp
        end;
skips:  end;


        for i:=in step 1 until n+m do
        ifa[Pt[i]+in]≠0 and vrow[i]≠ch2 then
begin
        if a[Pt[i]+0]>-0.000000001 and a[Pt[i]+in]<0 then goto skip;
        comp:=a[Pt[i]+0]/a[Pt[i]+in];
        ifcomp>-0.000000001 and comp-ratio<0.000000001 then
        begin
        if ratio-comp<0.000000001thennxt:= ratio;
        j:=i; ratio:=comp;
        end;
skip:       end;
        outgo:=j;
```

```
        if nxt-ratio<0.00000001 then  begin
        list[0]:=0;
        for i:= 1 step 1 until in do
                if a[Pt[in]+i]≠0  and vrow[i]≠ch2 then

        beginsgn:=ifvcol[i]=vrow[in]then-1 else 1;
        ifa[Pt[i]+0]>-0.000000001 and sgn*a[Pt[in]+i]<0 then goto kips;
                comp:=sgn*a[Pt[i]+0]/a[Pt[in]+i];
        if comp>-0.000000001 andcomp-ratio<0.000000001then

                beginlist[0]:=list[0]+1;
                      list[list[0]]:=i;
                end;
kips:   end;

        fori:=m+n step-1 until in+1 do
                ifa[Pt[i]+in]≠0and vrow[i]≠ch2
        then
                begin
                ifa[Pt[i]+0]>-0.000000001 anda[Pt[i]+in]<0 then goto kip;
        comp:=a[Pt[i]+0]/a[Pt[i]+in];
        ifcomp>-0.000000001and comp-ratio<0.000000001 then
        begin list[0]:=list[0]+1;
                list[list[0]]:=i;
        end;
kip:    end;

        j:=list[0]*random; \\\\\\\ifj=0 thenj:=1;
        outgo:=list[j];
end;

        if ratio>10000000 then begin
                if ch2=vrow[0] then goto unboundelse goto infeas
                                end;
end of procedure;
procedure duopivot(a,in,out,lmt,Pt,iter,vcol,vrow);value in,out,lmt;
        integer in,out,lmt,iter;integer array vcol,vrow,Pt;
        real array a;
        begin real array rhi,rlo,chi,clo,srhi,srlo,schi,sclo[0:lmt];
        integer i,j,lo,hi;real inversehilo,inverselohi,mplr1,mplr2;
        lo:=in;hi:=out;if lo>hi then beginlo:=hi;hi:=in end;
        for i:=0 step 1 until lmt do
begin
        if i<lo then beginrlo[i]:=a[Pt[lo]+i];rhi[i]:=a[Pt[hi]+i];
                clo[i]:=ifvrow[i]=vcol[lo]then-a[Pt[lo]+i]elsea[Pt[lo]\\+i];
                chi[i]:=ifvrow[i]=vcol[hi]then-a[Pt[hi]+i]elsea[Pt[hi]+i];
                      end;
        if i>lo andi< hi then begin
                rlo[i]:=ifvcol[lo]=vrow[i]then \\\\-a[Pt[i]+lo]elsea[Pt[i]+lo];
                rhi[i]:=a[Pt[hi]+i];clo[i]:=a[Pt[i]+lo];
                chi[i]:=ifvcol[hi]=vrow[i]then-a[Pt[hi]+i]\\elsea[Pt[hi]+i];
                                end;

        if i> hi then begin rlo[i]:=ifvcol[lo]=vrow[i]then\\-a[Pt[i]+lo]
                                                elsea[Pt[i]+lo];
```

```
                rhi[i]:=ifvcol[hi]=vrow[i]then-a[Pt[i]+hi] else a[Pt[i]+hi];
                clo[i]:=a[Pt[i]+lo];
                chi[i]:=a[Pt[i]+hi]\\;
                        end;
end;

        inversehilo:=1/rhi[lo];inverselohi:=1/clo[hi];
for i:=0 step 1 until lmt do
beginsrlo[i]:=rlo[i]-rhi[i]*clo[lo]*inversehilo;
        srhi[i]:=rhi[i]*inversehilo;
        sclo[i]:=-clo[i]*inverselohi;
        schi[i]:=chi[i]-clo[i]*rhi[hi]*inversehilo
end;
        srlo[lo]:=sclo[lo];schi[hi]:=srhi[hi];
        srhi[lo]:=sclo[hi]:=1/rhi[lo];


inverselohi:=1/schi[lo];
for j:=0 step 1 until lmt do
        if j≠hi or j≠ lo then

begin

        mplr1:=rhi[j]*inversehilo;mplr2:=srlo[j]*inverselohi;
                for i:= j step 1 until lmt do if i≠hi or i≠lo
        then
        a[Pt[i]+j]:=a[Pt[i]+j]-clo[i]*mplr1-schi[i]*mplr2
end;
\\
        for i:= 0  step 1 until lmt do

begin if i≠ hi then
        rhi[i]:=srhi[i]-srlo[i]*schi[hi]/schi[lo] else
        rhi[i]:=-srhi[i]/schi[lo];

        if i< lo then a[Pt[lo]+i]:=rhi[i] else
        a[Pt[i]+lo]:=if vcol[i]=vrow[lo] then rhi[i] else-rhi[i];

        ifi≠hi then
        rlo[i]:=srlo[i]/srlo[hi]else rlo[i]:=1/srlo[hi];
        if  i< hi then a[Pt[hi]+i]:=rlo[i] else
        a[Pt[i]+hi]:=if vcol[i]=vrow[hi] then  rlo[i] else-rlo[i];
end;
        a[Pt[lo]+lo]:=rhi[hi];a[Pt[hi]+hi]:=rlo[lo];a[Pt[hi]+lo]:=rlo[hi];
i:=vcol[lo];vcol[lo]:=vrow[lo];vrow[lo]:=i;
i:=vcol[hi];vcol[hi]:=vrow[hi];vrow[hi]:=i;
iter:=iter+i;
end of proc duopivot;
```

```
procedure parametrize(a,m,n,Pt,vcol,vrow,i,Pr,an);value m,n,i,Pr,\\\\an;
                integer m,n,i;
                integer arrayvcol,vrow,Pt; realPr,an;arraya;\\\\\\\\\

        comment an, the original parameter,Pr the new one,i its location
                in the zero column;
begin array b[0:m+n]; boolean basic;real mu,nu,mplr,alpha,beta;
        integer p;
        basic:= false;
        if i<n and vrow[i]≠vcol[0] then basic:= true;
        if basic then goto next;
        if i>n  and vrow[i]=vcol[0] then basic := true;
next:   if basic then

          begin
                mu:=an-\\a[Pt[i]+0\\];nu:=-1000000;
                a[Pt[i]+0]:=a[Pt[i]+0]-an+Pr;goto transfer
          end;


mplr:=an-Pr;
for p:=0 step  1 until  i do
                b[p]:=if vrow[p]=vcol[i] then-a[Pt[i]+p]
                        else a[Pt[i]+p];

        for p:= i+1 step 1 until  m+n do
        b[p]:=a[Pt[p]+i];
mu:=1000000000; nu:=-100000000;beta:=a[P\\t[i]+0];alpha:=a[Pt[i]+i];

for p:=1 step 1 until  m+n do

        begin
        ifb[p]>0  then
                        begin
                                ifa[Pt[p]+0]/b[p]<mu
                                then mu:=a[Pt[p]+0]/b[p]
                        end;
        ifb[p]<0   then
                        begin
                                ifa[Pt[p]+0]/b[p]>nu
                                then nu:=a[Pt[p]+0]\\/b[p]
                        end;


        a[Pt[p]+0]:=a[Pt[p]+0]-mplr*b[p]
                end;
mu:=an-mu;      nu:=an-nu;
ifabs(a[Pt[i]+i])<0.0000000001 \\\\then

        begin
                a[Pt[0]+0]:=a[Pt[0]+0]-mplr*2*b[0];goto transfer
```

```
        ond;

        beta:= beta/alpha;alpha:=1/alpha;
        a[Pt[0]+0]:=a[Pt[0]+0]-beta*beta/alpha+(beta-mplr)*(beta-mplr)/alpha;
transfor:

end of parametrize;

load;


ra:=34359738368;rc:=137438953472;rb:=68719476736;rd:=13543288579;re:=1/ra;


rep:    selectinput(1);selectoutput(1);
paperthrow;tot:=instr/2048;no:= read;writetext(<prob%no>);
print(no,4,0);newline\\(2);

siz\\e:=read;prnt:=readBoolean;m:=read;n:=read;t:=read;trlength:=read;
initials:=read;eps:=read;
comment m represents the no of constraints not considering the upper bound;
ub:=trlength*512;
 comment sets up the upper bound;
length:=(m+2*n+1)*(m+2*n+2\\)/2+(m+n)+2*n+1+3;
lim:=entier(ub/(length+1\\));

begin integer arrayPt,vcol,vrow\\[0:m+2*n],clist,cchar[0:n],avlist[0:lim];
        real array  store[1:ub],str,fc,cbest,\\qd,phis[1:n],away[0:m+2*n],\\\
                          a[0:(m+2*n+1)*(m+2*n+2)/2];
        integer i,j,k,l;
frillput(a,vcol,vrow,Pt,qd,fc,str,phis,avlist,cbest,cchar);

initialize:comment the following initializes the parameters;
clist[0]:=0;empty:= true;
        chooseprob:=read;choosevar:=read;newline(1);writetext(<HEURISTIC-RULE-%no
                                                                              );
        print((chooseprob-1)*2+choosevar,2,0);newline(1);
src:=false;cnt:=0;
iter:=0;Phno:=2;
max:=0;         mkr:=false;setup:=0;avlist[0]:=lim-1;trno:=0;
totpr:=0;

inital:\\Tst:=true;ch1:=vcol[0];ch2:=vrow[0];
        dual:=primal:=true;
        monitor(a,iter,0,n+m,vcol,vrow,Pt,clist);
        dual:=primal:=false;

Setparameter:   if instr/2048-tot>11000 then beginnewline(1);writetext(<///BADCA
                                                                    SE>);
                newline(1);\\\\\\\\\\\\\\\\\\\\          goto exhaustend;[redacted];

                mk:= true;
        if not dual then
                for i:=1 step 1 until n+m do
```

```
                ifvrow[i]=vrow[0]and \\a[Pt[i]+0]<\\\\-0.000000001  then mk:= false;
                dual:=mk;
        if dual then
                begin mk:= true;fori:= 1 step 1 until n+m do
                if vrow[i]=vcol[0] anda[Pt[i]+0]<-0.000000001 then mk:= false;
                primal:=mk;

        if primal then goto  soln;comment The above is solution exit;


ifch1≠vrow[0] then begin ch1\\:=vrow[0];ch2:=vcol[0] end;
        end;


if t>0 then  Phno:=1 else Phno:=if dual then2 else 3;


comment the following produces the pivot rules;
        if Phno>1 then goto rule2;
        rule1:  outgoing:=m+n;j:=random*(m+n);ifj<1 then j:=1;
        for \\\\\\\\\\\\\\\\incoming\\:=j step -1 until 1 do
        ifvcol[incoming]=vcol[0]anda[Pt[outgoing]+incoming]≠0 then goto double;
        forincoming:=j step 1 until n+m do
        if vcol[incoming]=vcol[0] \\\\and
        a[Pt[outgo\\ing]+incoming]≠0 then goto double;
        error: writetext(←pivot≠rule≠has≠gone≠wrong→);writetext(←j=→);
        print(j,5,0); newline(1); goto scin;

        rule2:incoming:= ifnot Tst then incoming else largest(ch2,a,Pt,n,m,vrow);
        outgoing:=outgo(a,Pt,ch2,vrow,vcol,incoming,m,n,unbound,infeas);



if incoming=outgoing then
        begin
                Tst:=true;compi:=incoming; gotosingle
        end;
        ifabs(a[Pt[outgoing]+outgoing])>0.000000001 then
        begin
                Tst:=false;compi:=outgoing;goto single
        end;
        double:

        Tst:= true;monitor(a,iter,0,m+n,vcol,vrow,Pt,clist);


                duopivot(a,incoming,outgoing,m+n,Pt,iter,vcol,vrow);
        goto Setparameter;
        single:                 monitor(a,iter,compi,m+n,vcol,vrow,Pt,clist);
                        pivot(a,compi,m+n,Pt,iter,vcol,vrow);
        goto Setparameter;
```

```
infeas:         goto search;
unbound:        writetext(⟨unbounded⟩);goto exhaust;
soln:   lamup:=evaluate(qd,fc,a,Pt,vrow,str);
        lamlower:= setup+a[Pt[0]+0];
        monitor(a,iter,0,n+m,vcol,vrow,Pt,clist);
        ifcnt=0then begin
        count:=instr/2048-tot;
        up:=lamup;low:=lamlower;
                writetext(⟨initial␣upper␣and␣lower␣b\\ounds⟩);
                print(lamup,8,4);print(lamlower,8,4);
                w\\ritetext(⟨no␣instructions␣to␣\\\\\first␣solution⟩);
        print(count,4,0);newline(1);
                cnt:=iter
                end;
if lamup>clamup  then goto  transfer ;
if lamlower -eps< clamup then  goto  search;

stores:         if clist[0]<n thensend(store,a,Pt,clist,avlist[0],vrow,avlist);
se\\arch:
        if mkr then
begin   fetch(store,a,Pt,clist,vrow,vcol,lim,exhaust,avlist);
        goto  init\\al;
end

        else
begin
        fetch(store,a,Pt,clist,vrow,vcol,chprob(avlist,store),exhaust,avlist);
        j:=chvar(clist,a,Pt,vrow,fc,phis);
        for i:= 0 step 1 until n+m do away[i]:=a[Pt[i]+0];
        parametrize(a,m,n,Pt,vcol,vrow,j,str[j],str[j]+fc[j]/phis[j]);
        clist[0]:=clist[0]+1;clist[clist[0]]:=j;
        setup:=setup-2*fc[j];
        send(store,a,Pt,clist,lim,vrow,avlist);
        setup:=setup+2*fc[j];

        for i:=0 step 1 until n+m do a[Pt[i]+0]:=away[i];
        parametrize(a,m,n,Pt,vcol,vrow,j+m,0,phis[j]);
        goto inital
end;

transfer:       clamup:=lamup;clamlower:=lamlower;
        fori:= 1 step 1 until n do

        begin
                cbest[i]:=a[Pt[i]+0];
                cchar[i]:=vrow[i]
        end;

if lamlower-eps<clamup then goto search;
goto stores;
```

```
exhaust:        writetext(←the␣tree␣is␣exhausted→);
                monitorbest(cbest,cchar,clist);

newline(1);writetext(←total␣no␣of␣problems␣investigated→);print(totpr,7,0);
newline(1);writetext(←maximum␣store␣used→);print(max,5,0);
newline(1);writetext(←total␣iterations␣in␣the␣tree␣search␣phase→);print(iter-cnt,
                                                                        6,0);
newline(1);writetext(←location␣of␣the␣optimum␣solution→);
print((clamup-low)/(up-low),2,5);


newline(1); write text(←last␣tableau␣was␣optimal→);newline(2);
        writetext(←computing␣time␣in␣multiples␣of␣initial␣solution→);
        print((instr/2048-tot-count)/count,2,3);
        newline(2);writetext(←logarithm␣of␣instruction␣count→);
        print(ln(instr/2048-tot-count),2,5);newline(2);
        writetext(←instruction␣count→);print(instr/2048-tot,5,0);newline(2);
        end of blok;

        if no>1 then goto rep;
        end of program;
***Z
```

Appendix 13. Input Format and the Data for the Fixed Charge Problems.

probno | problem no.= 0 if last problem;

size | monitor parameter; prnt | boolean variable to indicate intermediate printing..;

m | no. of constraints; n | no. of variables; t | no. of equality constraints;

trlength | no. of blocks of main store asked for; initial | block no. on magtape; eps | order of accuracy of bound;

c_j | linear non-negative cost terms;

q_j | quadratic, non-positive (taken with reverse sign) cost terms;

d_j | the fixed charge terms;

$\emptyset_j$ | the upper bounds;

type | boolean variable indicating type of input full or sparse;

block | no. of blocks of input = 1 for full matrix input;

[Matrix of constant terms and coefficients]

ch1 | x primal variable identifier; ch2 | v dual variable identifier;

feasible | type boolean indicates if a bound from a feasible solution is to be input;

chvar | indicates rule no. for choice of variable; chprob | rule no. for the choice of problem;

```
1
2       false
5       5       0
12      50      0.00001
19      10      15      18      8
-0.141  -0.050  -0.161  -0.074  -0.062
92      38      48      134     111
266     368     175     476     246
true
1

-218    -9      -6      -2      -1              0
-245    -10     0       0       0               -4
-283    0       0       0       -19             -13
-226    -8      -20     -7      -15             -12
-371    -14     -17     -3      -16             -11
X       U
false
1       1

2
2       false
5       5       0
12      50      0.00001
6       18      1       8       13
-0.040  -0.057  -0.026  -0.041  -0.069
199     59      26      160     39
282     59      75      386     352

true
1

-231    -10     0       0       -16             -2
-334    -20     0       0       -8              -7
-270    -11     -1      -6      -12             -9
-281    0       -17     0       -4              -18
-389    -3      0       -5      0               0
X       U
false
1       1

3
2       false
5       5       0
12      50      0.00001
18      11      9       15      13
-0.084  -0.134  -0.038  -0.097  -0.097
30      14      118     66      200
404     157     428     301     254

true
1
```

For a description of this format see page

-320	-17	-1	0	-12	-10
-346	0	-5	-2	-19	0
-374	-16	0	0	0	-4
-262	0	0	0	-9	0
-397	0	-15	0	0	-18

X U

false

1 1

4

2 false

5 5 0

12 50 0.00001

14 1 9 10 17

-0.132 -0.004 -0.048 -0.043 -0.362

79 170 30 87 54

201 442 354 430 91

true

1

-378	0	0	-9	0	-2
-296	0	-12	0	-7	0
-311	-11	-6	0	0	0
-304	0	0	-16	-8	-1
-288	0	0	-20	0	0

X U

false

1 1

5

2 false

5 5 0

12 50 0.00001

13 3 10 4 20

-0.017 -0.050 -0.041 -0.022 -0.087

144 99 107 72 79

201 105 442 352 430

true

1

-277	-15	0	0	0	-12
-285	-4	-17	0	-10	0
-323	0	-1	0	-13	0
-349	0	-16	0	0	-7
-288	0	-6	-2	-5	-9

X U

false

1 1

For a description of this format see page

```
6
2       false
5       5       0
12      50      0.00001
17      20      11      1       12
-0.036  -0.089  -0.020  -0.039  -0.269
76      42      14      67      33
88      430     105     486     83
true
1

-326    0       0       0       -15     0
-252    -17     0       -14     0       0
-241    0       0       -6      -9      -11
-362    0       -8      0       -13     0
-290    0       0       -5      -3      0
X       U
false
1       1
7
2       false
5       5       0
12      50      0.00001
14      19      2       13      6
-0.086  -0.077  -0.089  -0.056  -0.039
150     23      78      166     122
321     470     428     450     304
true
1

-356    -1      0       0       -3      0
-390    0       -9      -15     0       0
-387    -11     0       0       0       -17
-225    -14     0       -10     0       0
-324    0       -6      0       -16     0
X       U
false
1       1
8
2       false
8       8       0
16      50      0.00001
16      13      3       11
4       19      14      10
-0.152  -0.331  -0.070  -0.053
-0.047  -0.079  -0.073  -0.310
110     58      95      172
77      20      41      38
200     77      377     355
169     462     351     60

true
1
```

For a description of this format see page

```
1
-340    0       -10     0       0
        -1      -12     -13     -18
-357    0       0       -20     -2
        -14     0       -16     0
-289    -7      -15     0       -3
        -4      0       -17     -5
-378    -19     -6      -8      -9
        -11     0       -10     -8
-287    -9      0       0       -18
        -12     -6      -1      -5
-255    -20     -11     0       0
        -18     -11     -13     0

-376    -17     0       0       -16
        -3      0       -13     -2
-331    -15     0       0       -14
        -19     -7      -4      0
X       U
false
1       1
9
2       false
8       8       0
16      50      0.00001
19      4       13      15
11      12      10      8
-0.183  -0.059  -0.041  -0.072
-0.073  -0.085  -0.056  -0.098
21      190     140     75
167     138     173     116
187     396     474     210
296     262     333     154

true
1
-260    -15     -16     -2      -11
        -17     -7      0       -5
-355    -14     -20     -4      0
        0       0       -3      -10
-223    -9      0       -12     0
        -8      -13     -6      -18
-214    0       -19     0       0
        0       -1      -1      -13
-318    -11     -3      -4      0
        0       -6      -14     -15
-212    0       -18     0       -9
        0       0       -3      -7
```

For a description of this format see page

```
-387    -7      0       -5      -10
        -20     0       0       -12
-371    -16     -17     -8      0
        -19     0       0       -2
X       U
false
1       1
10
2       false
8       8       0
12      50      0.00001
7       9       11      5
3       2       4       15
-0.037  -0.037  -0.033  -0.162
-0.023  -0.074  -0.042  -0.068
28      88      169     144
82      74      148     104
404     125     242     178
238     491     183     410

true
1
-291    -1      -13     -11     -3
        -4      0       0       -6
-345    -14     -15     0       -18
        0       -9      -7      0
-279    -5      -10     0       0
        0       -12     -16     -17
-302    -8      0       -19     0
        0       -2      -14     -17
-315    0       -20     -8      -16
        -5      -10     -1      -13
-371    0       0       -2      0
        -9      0       0       -4

-233    -18     0       -3      -7
        0       0       -6      -19
-392    -15     -11     -9      -17
        -7      0       -4      -5
X       U
false
1       1
11
2       false
8       8       0
12      50      0.00001
14      12      9       17
7       15      13      11
-0.040  -0.057  -0.026  -0.07
-0.041  -0.017  -0.111  -0.098
183     147     144     195
99      137     159     124
318     147     144     364
258     376     166     422
```

For a description of this format see page

```
true
1
-346    0       0       0       0
        0       0       -7      -10
-381    0       0       0       0
        0       -12     -2      0
-366    0       0       -14     -15
        -18     0       -6      -9
-353    0       0       0       -8
        -16     0       -11     -1
-350    0       -3      -13     -5
        -4      -17     -19     0
-302    -14     0       0       -2
        0       0       -11     -6

-393    -6      0       0       -12
        0       -4      -9      -18
-337    -7      -16     0       -15
        -17     -3      -19     0
X       U
false
1       1
12
2       false
8       8       0
12      50      0.00001
17      9       11      13
12      19      16      15
-0.132  -0.004  -0.04   -0.36
-0.01   -0.05   -0.04   -0.02
177     175     85      70
181     172     100     137
224     430     434     484
382     264     167     243

true
1
-237    0       0       0       0
        0       0       0       -7
-355    0       -10     0       0
        0       0       0       0
-351    -12     -2      0       0
        0       0       -14     -15
-368    0       0       -6      0
        0       -9      0       0
-278    0       -8      -16     0
        0       0       -11     -1
-323    0       -3      -13     -5
        -4      -17     0       0
```

For a description of this format seepage

-307	0	-14	0	0
	-2	0	0	0
-228	-11	0	-6	0
	0	-12	0	-4
X	U			
false				
1	1			
13				
2	false			
8	8	0		
12	50	0.00001		
16	12	17	13	
19	11	9	7	
-0.36	-0.08	-0.02	-0.04	
-0.027	-0.09	-0.08	-0.06	
178	97	102	200	
78	167	142	196	
338	222	343	287	
141	300	159	366	
true				
1				
-386	0	0	-16	0
	0	0	-2	0
-267	0	0	-8	-3
	0	0	-5	0
-372	-1	0	-17	0
	0	-13	0	0
-390	-9	-11	-6	0
	0	0	0	0
-357	0	-8	0	0
	-11	0	0	-1
-376	0	0	-3	0
	0	0	0	0
-385	0	0	-17	-16
	-15	0	0	0
-303	0	-7	0	0
	0	0	0	-1
X	U			
false				
1	1			
14				
2	false			
8	8	0		
12	50	0.00001		
17	7	8	18	
10	13	20	14	
-0.05	-0.07	-0.33	-0.15	
-0.04	-0.079	-0.073	-0.31	
177	96	151	149	
106	51	135	118	
420	330	242	196	
148	112	360	229	

For a description of this format see page

true
1
-325 0 -14 0 0
-2 0 0 0
-394 -11 0 -6 0
0 -12 0 -4
-387 -9 0 -7 -16
0 0 -15 -17
-368 -3 0 0 0
-10 0 -8 0
-227 0 0 0 0
0 0 0 -5
-324 0 0 0 0
0 0 -13 0

-277 -1 -14 0 -8
0 0 -10 0
-348 0 0 -9 0
-13 0 0 -11
X U
false
1 1

15
2 false
10 10 0
12 50 0.00001
19 10 15 18 17
8 5 9 12 16
-0.037 -0.037 -0.033 -0.162 -0.023
-0.07 -0.04 -0.07 -0.033 -0.068
177 96 151 122 149
166 135 118 79 94
475 365 447 351 234
345 274 237 171 435
true
1
-333 0 0 0 -5 -4
-14 -11 -1 -6 -16
-317 -8 0 -7 -3 0
0 -2 0 -12 0
-390 0 0 0 -19 -17
-13 -10 -15 -18 -9
-259 0 0 -3 0 -7
-14 -15 -6 -5 0
-327 0 0 0 0 -18
-10 -4 -16 -19 -1
-336 -13 -9 -8 0 -17
-11 -12 0 0 -2
-397 -18 -11 -13 -2 -8
-9 -5 -14 -19 -12
-223 0 0 -17 0 -3
-1 0 0 -10 -4
-323 -6 -15 0 -16 0
0 -7 0 0 0
-383 -14 -3 -12 0 0
-10 -4 -6 -1 -19
X U
false
1 1

For a description of this format see page

```
16
2       false
10      10      0
12      50      0.00001
16      8       18      4       14
20      11      9       7       13
-0.035  -0.026  -0.061  -0.076  -0.061
-0.099  -0.07   -0.07   -0.07   -0.07
150     105     173     62      153
90      101     191     115     172
150     479     225     461     314
139     429     343     282     238

true
1
-350    -9      0       -1      -15     -16
        -2      0       -13     0       0
-305    -18     0       0       -14     -4
        0       -8      0       0       -12
-373    -3      0       -6      -19     -11
        -7      0       -10     0       0
-257    0       -17     0       0       0
        0       0       -5      0       -1
-267    0       0       0       0       -13
        0       -3      -11     -8      0
-329    -16     0       0       0       -15
        -9      0       0       0       0

-363    0       -18     -5      -10     -17
        -19     0       -14     0       0
-312    0       0       -2      -12     -4
        -7      -6      0       -9      0
-228    -10     -3      0       0       0
        -17     0       -2      -16     -5
-279    0       -1      0       0       0
        -7      0       0       -13     -15
X       U
false
1       1
17
2       false
10      10      0
12      50      0.00001
16      11      7       14      18
17      15      10      20      19
-0.058  -0.076  -0.045  -0.017  -0.067
-0.085  -0.054  -0.012  -0.071  -0.012
164     115     80      162     113
110     136     181     191     137
420     280     400     236     163
283     221     431     285     209
```

For a description of this format see page

```
true
1
-298    -14     0       0       -8      0
        0       0       0       0       0
-394    0       -11     -17     0       0
        0       -2      0       0       0
-391    -16     0       0       0       0
        0       -9      -4      -7      -10
-244    0       -15     0       -5      0
        0       -12     0       0       0
-269    -13     0       0       0       -1
        0       0       0       -3      -6
-301    -2      0       -8      0       -9
        -11     0       0       0       0

-348    -17     0       0       0       0
        0       0       -6      0       0
-373    -16     0       0       -12     -3
        -13     -7      0       -4      -15
-382    0       0       0       -10     0
        0       0       0       0       0
-364    0       0       -1      0       0
        -5      0       -14     0       -11
X       U
false
1       1

18
12      false
5       5       0
12      50      0.00001
8       6       4       2       0.00001
0       0       0       0       0
300     300     300     300     300
500     410     285     145     45
true

1
-90     -1   0   0   0   0
-215    -1   -1    0    0    0
-355    -1    -1     -1     0    0
-455    -1   -1   -1   -1   0
-500    -1    -1     -1     -1     -1
X       U
false
1       1
```

For a description of this format see page

```
19
12      false
6       6       0
12      50      0.00001
10   8   6   4   2   0.0001
0   0   0   0   0   0
300  300  300  300  300  300
560  470  345  205  105  60
true
1
-90     -1   0   0   0   0   0
-215    -1   -1   0   0   0   0
-355    -1   -1   -1   0   0   0
-455    -1   -1   -1   -1   0   0
-500    -1   -1   -1   -1   -1   0
-560    -1   -1   -1   -1   -1   -1
X       U
false
1       1

20
12      false
7       7       0
12      50      0.00001
12   10   8   6   4   2   0.0001
0   0   0   0   0   0   0
300  300  300  300  300  300  300
690  600  475  335  235  190  130
true
1
-90     -1   0   0   0   0   0   0
-215    -1   -1   0   0   0   0   0
-355    -1   -1   -1   0   0   0   0
-455    -1   -1   -1   -1   0   0   0
-500    -1   -1   -1   -1   -1   0   0
-560    -1   -1   -1   -1   -1   -1   0
-690    -1   -1   -1   -1   -1   -1   -1
X       U
false
1       1
21
8       false
8       8       0
12      50      0.00001
14   12   10   8   6   4   2   0.0001
0   0   0   0   0   0   0   0
300  300  300  300  300  300  300  300
840  750  625  485  385  340  280  150

true
1
-90     -1   0   0   0   0   0   0   0
-215    -1   -1   0   0   0   0   0   0
-355    -1   -1   -1   0   0   0   0   0
-455    -1   -1   -1   -1   0   0   0   0
-500    -1   -1   -1   -1   -1   0   0   0
-560    -1   -1   -1   -1   -1   -1   0   0
-690    -1   -1   -1   -1   -1   -1   -1   0
-840    -1   -1   -1   -1   -1   -1   -1   -1
X       U
```

For a description of this format see page

```
false
1        1
22
8        false
9        9        0
12       50       0.00001
16   14   12   10   8   6   4   2   0.0001
0   0   0   0   0   0   0   0   0
300   300   300   300   300   300   300   300   300
900   810   685   545   445   400   340   210   60
```

```
true
1
-90      -1   0   0   0   0   0   0   0   0
-215     -1   -1   0   0   0   0   0   0   0
-355     -1   -1   -1   0   0   0   0   0   0
-455     -1   -1   -1   -1   0   0   0   0   0
-500     -1   -1   -1   -1   -1   0   0   0   0
-560     -1   -1   -1   -1   -1   -1   0   0   0
-690     -1   -1   -1   -1   -1   -1   -1   0   0
-840     -1   -1   -1   -1   -1   -1   -1   -1   0
-900     -1   -1   -1   -1   -1   -1   -1   -1   -1
X        U
false
1        1
23
12       false
5        5        0
12       50       0.00001
8   6   4   2   0.00001
0   0   0   0   0
250   250   250   250   250
455   375   275   150   50
true
1
-80      -1   0   0   0   0
-180     -1   -1   0   0   0
-305     -1   -1   -1   0   0
-405     -1   -1   -1   -1   0
-455     -1   -1   -1   -1   -1
X        U
false
1        1
24
6        false
6        6        0
12       50       0.00001
10   8   6   4   2   0.00001
0   0   0   0   0   0
250   250   250   250   250   250
505   426   325   200   100   50
```

For a description of this format see page

```
true
1
-80     -1   0   0   0   0   0
-180    -1   -1   0   0   0   0
-305    -1   -1   -1   0   0   0
-405    -1   -1   -1   -1   0   0
-455    -1   -1   -1   -1   -1   0
-505    -1   -1   -1   -1   -1   -1
X       U
false
1       1
25
6       false
7       7       0
12      50      0.00001
12   10   8   6   4   2   0.000001
0   0   0   0   0   0   0
250   250   250   250   250   250   250
605   525   425   300   200   150   100
true
1
-80     -1   0   0   0   0   0   0
-180    -1   -1   0   0   0   0   0
-305    -1   -1   -1   0   0   0   0
-405    -1   -1   -1   -1   0   0   0
-455    -1   -1   -1   -1   -1   0   0
-505    -1   -1   -1   -1   -1   -1   0
-605    -1   -1   -1   -1   -1   -1   -1
X       U
false
1       1

26
6       false
8       8       0
12      50      0.00001
14   12   10   8   6   4   2   0.00001
0   0   0   0   0   0   0   0
250   250   250   250   250   250   250   250
730   650   550   425   325   275   225   125
true
1
-80     -1   0   0   0   0   0   0   0
-180    -1   -1   0   0   0   0   0   0
-305    -1   -1   -1   0   0   0   0   0
-405    -1   -1   -1   -1   0   0   0   0
-455    -1   -1   -1   -1   -1   0   0   0
-505    -1   -1   -1   -1   -1   -1   0   0
-605    -1   -1   -1   -1   -1   -1   -1   0
-730    -1   -1   -1   -1   -1   -1   -1   -1
X       U
false
1       1
```

For a description of this format see page

```
27
6        false
9        9        0
12       50       0.00001
16   14   12   10   8   6   4   2   0.00001
0    0    0    0    0   0   0   0   0
250  250  250  250  250 250 250 250 250
830  750  650  525  425 375 325 225 100
true
1
-80      -1   0   0   0   0   0   0   0   0
-180     -1   -1  0   0   0   0   0   0   0
-305     -1   -1  -1  0   0   0   0   0   0
-405     -1   -1  -1  -1  0   0   0   0   0
-455     -1   -1  -1  -1  -1  0   0   0   0
-505     -1   -1  -1  -1  -1  -1  0   0   0
-605     -1   -1  -1  -1  -1  -1  -1  0   0
-730     -1   -1  -1  -1  -1  -1  -1  -1  0
-830     -1   -1  -1  -1  -1  -1  -1  -1  -1
X        U
false

1        1
```

For adescription of this format see page

RANDOM LINEAR PROBLEMS
Problem Solutions in terms of x-variables only

NO	x1=	x2=	x3=	x4=	x5=	x6=	x7=	x8=	x9=	x10=
1	24.2222	0.0000	0.0000	0.0000	21.7692					
2	19.9778	0.0000	65.8133	0.0000	15.6111					
3	23.3750	26.4667	0.0000	29.1111	0.0000					
4	0.0000	51.8333	42.0000	0.0000	0.0000					
5	18.4667	29.1644	0.0000	22.6027	0.0000					
6	0.0000	0.0000	18.0000	66.6667	0.0000					
7	0.0000	0.0000	26.0000	118.6667	22.7647					
8	0.0000	0.0000	0.0000	0.0000	21.8774	0.0000	24.4710	0.0000		
9	0.0000	18.0294	0.0000	0.0000	0.0000	0.0000	0.0000	32.2500		
10	18.8730	0.0000	0.0000	0.0000	68.0318	0.0000	11.5397	0.0000		
11	0.0000	0.0000	0.0000	0.0000	0.0000	21.5833	61.0000	0.0000		
12	0.0000	35.5000	61.3333	0.0000	0.0000	0.0000	0.0000	33.8571		
13	0.0000	44.6250	125.3333	0.0000	0.0000	0.0000	0.0000	0.0000		
14	0.0000	20.8055	0.0000	0.0000	16.8615	17.7000	24.9231	45.4000		
15	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	46.9894	0.0000	18.5851	9.2872
16	0.0000	15.1177	0.0000	15.5191	21.9333	0.0000	0.0000	0.0000	20.2986	0.0000
17	20.6923	3.5333	32.4519	38.2000	0.0000	0.0000	0.0000	23.6820	0.0000	0.0000

PRODUCTION SCHEDULING PROBLEMS

Problem Solutions in terms of x variables only

NO	x1=	x2=	x3=	x4=	x5=	x6=	x7=	x8=	x9=	x10=
18	215.00	0.00	285.00	0.00	0.00					
19	215.00	0.00	240.00	0.00	105.00	0.00				
20	215.00	0.00	240.00	0.00	105.00	0.00	130.00			
21	215.00	0.00	240.00	0.00	105.00	0.00	280.00	0.00		
22	215.00	0.00	240.00	0.00	105.00	0.00	130.00	210.00	0.00	
23	180.00	0.00	125.00	150.00	0.00					
24	180.00	0.00	125.00	200.00	0.00	0.00				
25 *	180.00	0.00	125.00	150.00	0.00	150.00	0.00			
26	180.00	0.00	225.00	0.00	100.00	0.00	225.00	0.00		
27	180.00	0.00	225.00	0.00	100.00	0.00	100.00	225.00	0.00	

* There are multiple optima in this set of problems.

RANDOM QUADRATIC PROBLEMS

Problem Solution in terms of x-variables only

NO	x1=	x2=	x3=	x4=	x5=	x6=	x7=	x8=	x9=	x10=
1	15.7923	12.6449	0.0000	0.0000	21.7692					
2	19.9778	0.0000	65.8133	0.0000	15.6111					
3	17.8611	0.0000	0.0000	29.1111	22.0556					
4	0.0000	51.8333	42.0000	0.0000	0.0000					
5	13.4667	29.1644	0.0000	22.6027	0.0000					
6	0.0000	18.0000	0.0000	66.6667	0.0000					
7	0.0000	0.0000	26.0000	118.667	22.7647					
8	0.0000	0.0000	0.0000	0.0000	21.8774	0.0000	27.4710	0.0000		
9	0.0000	18.0294	0.0000	0.0000	0.0000	0.0000	0.0000	32.2500		
10	7.1078	9.1541	0.0000	0.0000	41.2111	0.0000	17.5098	0.0000		
11	0.0000	0.0000	0.0000	0.0000	3.4590	27.4934	25.5394	16.7224		
12	0.0000	35.5000	61.3333	0.0000	0.0000	0.0000	0.0000	33.8571		
13	0.0000	44.6250	125.3333	0.0000	0.0000	0.0000	0.0000	0.0000		
14	0.0000	20.8055	0.0000	0.0000	16.8615	17.7000	24.9231	45.4000		
15	0.0000	0.0000	15.7404	0.0000	0.0000	0.0000	46.1428	0.0000	9.5443	0.0000
16	0.0000	15.1177	0.0000	15.5191	21.9333	0.0000	0.0000	0.0000	20.2986	0.0000
17	20.6923	3.5333	32.4519	38.2000	0.0000	0.0000	0.0000	23.6820	0.0000	0.0000

APPENDIX 14

DATA FOR A FIXED CHARGE PROBLEM FORMULATED AS AN INTEGER QUADRATIC PROGRAMMING PROBLEM.

```
DATA
SYM-DATA
8
15      10      0       5       0.00001
-144    -99     -107    -72     -79
-13     -3      -10     -4      -20

0
0   0
0   0   0
0   0   0   0
0   0   0   0   0
0   0   0   0   0   -0.017
0   0   0   0   0   0       -0.050
0   0   0   0   0   0         0     -0.041
0   0   0   0   0   0         0       0       -0.022
0   0   0   0   0   0           0       0       0       -0.087
1   1   0   0   0   0   0   0   0   0   0
1   0   1   0   0   0   0   0   0   0   0
1   0   0   1   0   0   0   0   0   0   0
1   0   0   0   1   0   0   0   0   0   0
1   0   0   0   0   1   0   0   0   0   0

0   -201    0   0   0   0   1   0   0   0   0
0   0   -105   0   0   0   0   1   0   0   0
0   0   0   -442    0   0   0   0   1   0   0
0   0   0   0   -352    0   0   0   0   1   0
0   0   0   0   0   -430    0   0   0   0   1
-277    0   0   0   0   0
-15     0       0       0       -12
-285    0   0   0   0   0
-4      -17     0       -10     0
-323    0   0   0   0   0
0       -1      0       -13     0
-349    0       0       0       0               0
0       -16     0       0       -7
-288    0   0   0   0   0
9       -6      -2      -5      -9
X       U
***Z
```