APPLICATION OF LINEAR PSEUDO-BOOLEAN PROGRAMMING TO COMBINATORIAL PROBLEMS

by /2 4/

NADIPURAM SREERANGA CHAR ANANTHA RANGA CHAR

B.E. (Mech.), University of Mysore, India, 1965

A MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Industrial Engineering

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1970

Approved by:

[Signature]

Major Professor
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGEMENT</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
<tr>
<td><strong>CHAPTER I. INTRODUCTION</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Zero-one Linear Programming</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Proposed Research</td>
<td>6</td>
</tr>
<tr>
<td><strong>CHAPTER II. LINEAR PSEUDO-BOOLEAN ALGORITHM</strong></td>
<td>8</td>
</tr>
<tr>
<td>2.1 Basic Concepts</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Sample Problem</td>
<td>15</td>
</tr>
<tr>
<td>2.3 Computational Algorithm</td>
<td>20</td>
</tr>
<tr>
<td><strong>CHAPTER III. APPLICATION TO COMBINATORIAL PROBLEMS</strong></td>
<td>24</td>
</tr>
<tr>
<td>3.1 Shop Scheduling Problem</td>
<td>24</td>
</tr>
<tr>
<td>3.2 Assembly-Line Balancing Problem</td>
<td>31</td>
</tr>
<tr>
<td>3.3 Delivery Problem</td>
<td>35</td>
</tr>
<tr>
<td>3.4 Travelling Salesman Problem</td>
<td>39</td>
</tr>
<tr>
<td>3.5 Capital Allocation Problem</td>
<td>43</td>
</tr>
<tr>
<td>3.6 Fixed-Charge Problem</td>
<td>46</td>
</tr>
<tr>
<td><strong>CHAPTER IV. COMPUTATIONAL EXPERIENCE</strong></td>
<td>50</td>
</tr>
<tr>
<td>4.1 Results of the Pseudo-Boolean Algorithm</td>
<td>50</td>
</tr>
<tr>
<td>4.2 Computational difficulties</td>
<td>53</td>
</tr>
<tr>
<td><strong>CHAPTER V. SUMMARY AND CONCLUSIONS</strong></td>
<td>60</td>
</tr>
<tr>
<td><strong>REFERENCES</strong></td>
<td>63</td>
</tr>
</tbody>
</table>
THIS BOOK CONTAINS NUMEROUS PAGES WITH ILLEGIBLE PAGE NUMBERS THAT ARE CUT OFF OR MISSING.

THIS IS AS RECEIVED FROM THE CUSTOMER.
APPENDIX A. CONVERSION OF INTEGER PROGRAMMING TO A ZERO-ONE FORM

A.1 The Simple Expansion Technique 68

A.2 The Balas Binary Device 69

APPENDIX B. COMPUTER PROGRAM LISTING 72
ACKNOWLEDGEMENT

The author is deeply indebted to his major advisor, Dr. Said Ashour, for his guidance and assistance in the completion of this thesis. Thanks are extended to Prof. J. J. Smaltz for his help during the initial stages of computer programming.

The author wishes to thank Mr. Srinivasan Prakash for his invaluable help in correcting the manuscript and Mrs. Marie Jirak for her excellent work in typing this thesis.
LIST OF TABLES

Table 2.1 Equality constraints 10
Table 2.2 Inequality constraints 12
Table 2.3 Preferential order 13
Table 4.1 Computational Results for Scheduling Problems 54
Table 4.2 Computational Results for Line-Balancing Problems 55
Table 4.3 Computational Results for Delivery Problems 56
Table 4.4 Computational Results for Travelling Salesman Problems 57
Table 4.5 Computational Results for Capital Allocation Problems 58
Table 4.6 Computational Results for Fixed-Charge Problems 59
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2.1</td>
<td>Branching Tree for Sample Problem</td>
<td>20</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>Gantt Chart for a (2x3) Flow-Shop Sample Problem</td>
<td>30</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>Ordering position for Sample Problem</td>
<td>33</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>Delivery Routes for Sample Problem</td>
<td>37</td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION

The combinatorial problem is concerned with the study of the arrangement of elements into sets. The elements are usually finite in number, and the arrangement is restricted by certain boundary conditions imposed by the particular problem under investigation [53]. Most combinatorial problems can be classified into four types. In the first, the existence of the particular arrangement is unknown and the problem is to find whether the particular arrangement exists or not. These are called existence problems. In the second, the existence of the arrangement is known and the problem is to find that arrangement. These are called construction or evaluation problems. Finding all the possible arrangements comes under the third type which are known as enumeration problems. When it is necessary to choose the best combination defined using some criteria, the problems fall under the fourth type. These are known as extremization problems. Most of the combinatorial problems are one of these types, although the distinction is not always precise [7].

Various combinatorial problems such as shop scheduling, assembly-line balancing, delivery, travelling salesman, capital allocation and fixed-charge problems come under the category of extremization problems. In these problems, a given objective is to be optimized subject to a set of constraints arising due to the characteristics of the problem. Because the number of combinations increases non-linearly, direct
search is not practically feasible except for very small problems. Hence methods have to be devised to limit the search to a smaller subset of all solutions. In real situations, all the elements are integers and therefore the solution obtained must be integer-valued. Thus these problems can be formulated as integer programming problems so that the results are integers. By the proper utilization of zero-one variables, these problems can be converted into a zero-one program and can be solved by using the pseudo-Boolean program.

1.1 Zero-one Linear Programming

The integer linear programming problem may be stated as minimize
\[ c_1 x_1 + c_2 x_2 + \cdots + c_n x_n \]
subject to
\[ a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n \geq P_1 \]
\[ a_{21} x_1 + a_{22} x_2 + \cdots + a_{2n} x_n \geq P_2 \]
\[ \vdots \]
\[ a_{m1} x_1 + a_{m2} x_2 + \cdots + a_{mn} x_n \geq P_m \]
and
\[ x_j \geq 0, \quad j = 1, 2, \ldots, n, \]
where
\[ x_j \quad \text{denotes the } j^{\text{th}} \text{ unknown integer valued variable,} \]
\[ c_j \quad \text{denotes the unit cost of the variable } x_j, \]
\[ P_i \quad \text{denotes the level of } i^{\text{th}} \text{ requirement,} \]
\( a_{ij} \) denotes the number of units of \( x_j \) which satisfies the requirement \( p_i \).

In addition, if the value of \( x \) is limited to either zero or one, the zero-one linear programming problem is obtained. Any integer linear programming problem can usually be converted into zero-one linear programming problem by using either simple expansion technique [17] or Balas binary device [51] discussed in Appendix A.

The solution of linear programming without any integer restriction has been obtained by Dantzig [15] in collaboration with Giesler, Orden, Wood among others. In certain types of problems, a linear program without any integer constraints will have integer-valued solution. This occurs in the class of mathematically equivalent problems which include the assignment, transportation and network flow problems. Dantzig [17] has pointed out that the transportation problem, and hence this class of problems, always has integer solutions, given integer-valued demands and supplies.

If the solution of a linear programming problem does not have the required integer property then integer constraints have to be incorporated. Numerous algorithms have been proposed for the solution of general integer linear programming starting with those of Gomory [28,29] and Land and Doig [41]. These algorithms can be broadly divided into four classes according to the method employed:

1. Algebraic Approach
2. Combinatorial Approach
3. Enumerative Approach
4. Heuristic Approach
First, the algebraic approach is based on methods which generate new constraints, called cuts or cutting planes so as to restrict the solution space without eliminating any feasible integer points. Second, combinatorial approach is the method which is combinatorial in nature for which algebraic rather than exponential bounds are available for the number of steps required to solve a problem. Third, enumerative approach is the method of search over all possible solutions which limit the extent of search. Finally, heuristic approach refers to collection of heuristic rules for obtaining local optimal solutions utilizing computers.

The history of integer linear programming started with the significant contribution of Gomory [28,29]. Reviews by Beale [6] and Balinsky [3,4] provide an excellent coverage of the available literature. These reviews cover various important algorithms classified according to the above outlined scheme.

**Algebraic Approach.** The basic idea is that of successively deducing supplementary linear constraints, until a new linear program, whose solution satisfies the integer requirements, is obtained. New constraints, called cuts or cutting planes are generated so as to restrict the solution space without eliminating any feasible integer points. Gomory [28,29,31] has proposed this approach for solving a pure integer problem in which all variables are required to be integer-valued. Gomory [30] has generalized his method for the problem where \( a_{ij} \) is integer-valued. Young [59] has developed a primal integer programming algorithm which he simplified later [60]. Glover has worked on the cuts proposed by Gomory [26] and Ben-Israel and Charnes [27] with a view to developing a general class of cuts. However Glover
has not been successful in embodying these cuts in an efficient algorithm.

**Combinatorial Approach.** For integer programming problems which are not of transportation type, very few examples of this approach exist. There are two main instances. One due to Gomory [28] is a combinatorial, recursive procedure for obtaining an optimal solution to the asymptotic problem. The computation proceeds on the finite group, G, and an algebraic upper bound on the number of steps necessary to obtain an optimal solution is known as a priori. The other main instance is concerned with integer programming problems related to graphs. Edmonds [19] has been the first to develop an algorithm on these lines for simple matching problems. Balinsky [4] has improved the work of Edmonds by reducing the storage requirements. Edmonds, Johnson and Lockhart [40] made further progress in simple matching and covering problems.

**Enumerative Approach.** This can be broadly classified into two subclasses: (1) single-branch search which is exemplified by the method of Balas [1] for zero-one problem; and (2) multi-branch search which is exemplified by the methods of Land and Doig [41] for the mixed integer problem and Little et al. [44] for the travelling salesman problem. Because of the large computer memory required, relatively little computational experience has been reported regarding multi-branch schemes.

Algorithms belonging to single-branch search are used primarily to solve zero-one linear programming problems. Two of the successful algorithms are the additive algorithm of Balas [1] and the multiphase

Heuristic Approach. These methods involve either solution of one or a sequence of derived problems or the use of some heuristics or reasonable rules for finding a local optimum. The notable research on this approach was done by Lin [45] who obtained approximate solutions to travelling salesman problems. Some of the algorithms have been computationally inefficient or made use of certain problem characteristics. These include the Boolean algebra approaches used by Fortet [21] and Camion [13] and a dynamic programming approach proposed by Glass [24] and refined by Rao [52]. The latter suffers from the dimensionality difficulty.

1.2 Proposed Research

This thesis makes use of an algorithm proposed by Hammer and Rudeanu [35] in solving the zero-one programming problems. Briefly, the algorithm utilizes a branching and bounding procedure using a set of rules. These rules are due to the properties of pseudo-Boolean functions. A systematic procedure in applying the rules will result in obtaining the optimal solution.

The basic approach of the pseudo-Boolean algorithm is discussed in Chapter II. The fundamental concepts of the algorithm are discussed and a sample problem is presented for illustration. A computer
program is written in Fortran IV for IBM 360/50 computer. Details
of the computer program are shown in Appendix B.

The combinatorial problems, namely, shop scheduling, line
balancing, delivery, travelling salesman, capital allocation, and
fixed-charge problems are formulated as zero-one linear programming
problems in Chapter III. A sample problem for each type is presented
and the results discussed.

Several problems have been solved and the computational results
are reported in Chapter IV. Conclusions are given in Chapter V.
CHAPTER II

LINEAR PSEUDO-BOOLEAN ALGORITHM

In the late forties, the theory of Boolean Algebra has been first applied in the study of the switching circuits. This is due to the fact that each element of the switching circuit can be either in "ON" or in "OFF" condition, and thus they can be easily represented by using zero-one variables. Since then the use of zero-one variables to represent binary decisions became a general practice. Binary decision problems are frequently found in the theory of graphs, combinatorial and other discrete optimization problems.

Pseudo-Boolean programming, a method for solving zero-one programs, has been developed by Rosenberg et al. [36] using a method proposed by Fortet [21]. The present algorithm has been developed by Hammer and Rudeanu [35] using the principle of dynamic programming and Boolean procedures. This chapter is devoted to the discussion and illustration of the linear pseudo-Boolean algorithm. A computer program has been written in FORTRAN IV for IBM 360/50 computer. The listing of the program with a sample problem is shown in Appendix B.

2.1 Basic Concepts

The approach used in this thesis is based on properties of Pseudo-Boolean functions. A pseudo-Boolean function may be defined as a real-valued function \( f(x_1, x_2, \ldots, x_n) \) with zero-one variables. An equation (or inequality) involving only pseudo-Boolean functions on both sides, is called a pseudo-Boolean equation (or inequality). A pseudo-Boolean program is a procedure to optimize a pseudo-Boolean
function. The variables involved may be either unrestricted or sub-
ject to constraints expressed by a system of pseudo-Boolean equalities
and inequalities. Whenever the function and the constraints are
linear, the problem reduces to linear pseudo-Boolean programming.

The method utilizes branching procedure and is categorized as
an enumerative and testing technique. It uses a set of rules
dependent on the properties of pseudo-Boolean functions. The method
limits the number of branches to be investigated to a smaller subset.
Incorporating a bounding technique with the objective function,
the search converges to the optimal value rapidly. Improved results
at each successive trial are utilized to improve the convergence.

The linear pseudo-Boolean programming may be stated as follows: -

Minimize
\[ a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq P_1 \]

subject to
\[ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \geq P_2 \]
\[ a_{31}x_1 + a_{32}x_2 + \ldots + a_{3n}x_n \geq P_3 \]
\[ \vdots \]
\[ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \geq P_m \]

where
\[ x_j = 0 \text{ or } 1, \quad j = 1, 2, \ldots, n \]

and
\[ P_1 = \text{upper bound of the objective function.} \]

The properties of pseudo-Boolean equations and inequalities are
shown in Tables 2.1 and 2.2. To obtain the solution to the problem,
### Table 2.1 Equality Constraints

<table>
<thead>
<tr>
<th>No.</th>
<th>Case</th>
<th>Conclusions</th>
<th>Fixed Variables</th>
<th>Remaining Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$P_i &lt; 0$</td>
<td>No solutions</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>$P_i = 0$</td>
<td>All of appearing variables fixed</td>
<td>$x_1 = x_2 = \ldots = x_m = 0$</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>$P_i &gt; 0$ and $a_{i1} \geq \cdots \geq a_{ip} &gt; a_{i(p+1)} \geq \cdots \geq a_{im}$</td>
<td>Part of appearing variables fixed</td>
<td>$x_1 = \ldots = x_p = 0$</td>
<td>$\sum_{j=p+1}^{m} a_{ij} x_j = P_i$</td>
</tr>
<tr>
<td>4</td>
<td>$P_i &gt; 0$ and $a_{i1} = \ldots = a_{ip} = a_{i(p+1)} &gt; \cdots &gt; a_{im}$</td>
<td>There are $(p+1)$ possibilities</td>
<td>$x_1 = \ldots = x_{(k-1)} = x_{(k+1)} = \ldots = x_m = 0$, $k=1,2,\ldots,p.$</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\alpha: x_k = 1,$</td>
<td>$\beta: x_1 = \ldots = x_p = 0$</td>
<td>$\sum_{j=p+1}^{m} a_{ij} x_j = P_i$</td>
</tr>
<tr>
<td>5</td>
<td>$P_i &gt; 0$, $a_{ij} &lt; P_i$ ($j=1,\ldots,m$) and $\sum_{j=1}^{m} a_{ij} &lt; P_i$</td>
<td>No solutions</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>$P_i &gt; 0$, $a_{ij} &lt; P_i$ ($j=1,\ldots,m$) and $\sum_{j=1}^{m} a_{ij} = P_i$</td>
<td>All of appearing variables fixed</td>
<td>$x_1 = \ldots = x_m = 1$</td>
<td>-</td>
</tr>
<tr>
<td>No.</td>
<td>Case</td>
<td>Conclusions</td>
<td>Fixed Variables</td>
<td>Remaining Equation</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>-------------</td>
<td>-----------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>7</td>
<td>$p_i &gt; 0, a_{ij} &lt; p_i$ (j=1,...,m)</td>
<td>One variable fixed</td>
<td>$x_1 = 1$</td>
<td>$\sum_{j=2}^{m} a_{ij} x_j = p_i - a_{11}$</td>
</tr>
<tr>
<td></td>
<td>$\sum_{j=1}^{m} a_{ij} &lt; p_i$ and $\sum_{j=2}^{m} a_{ij} &lt; p_i$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>$p_i &gt; 0, a_{ij} &lt; p_i$ (j=1,...,m)</td>
<td>There are two possibilities</td>
<td>$\gamma_1: x_i = 1$</td>
<td>$\sum_{j=2}^{m} a_{ij} x_j = p_i - a_{11}$</td>
</tr>
<tr>
<td></td>
<td>$\sum_{j=1}^{m} a_{ij} &gt; p_i$ and $\sum_{j=2}^{m} a_{ij} &gt; p_i$</td>
<td>$\gamma_1, \gamma_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\gamma_2: x_1 = 0$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$\sum_{j=2}^{m} a_{ij} x_j = p_i$</td>
</tr>
<tr>
<td>No.</td>
<td>Case</td>
<td>Information</td>
<td>Conclusions</td>
<td>Fixed Variables</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>-------------</td>
<td>-------------</td>
<td>----------------</td>
</tr>
<tr>
<td>1</td>
<td>$P_i &lt; 0$</td>
<td>Redundant inequality</td>
<td>$\alpha_k: x_1=x_2=\ldots=x_{k-1}=0$, $x_k=1$ (k=1,2,\ldots,p)</td>
<td>$\beta: x_1=x_2=\ldots=x_p=0$</td>
</tr>
<tr>
<td>2</td>
<td>$P_i &gt; 0$ and $a_{i1} &gt; \ldots &gt; a_{ip} &gt; P_i &gt; a_{i(p+1)} &gt; \ldots &gt; a_{im}$</td>
<td>There are p+1 possibilities</td>
<td>$\alpha_1.\alpha_2.\ldots,\alpha_p.\beta$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$P_i &gt; 0, a_{ij} &lt; P_i$ (j=1,2,\ldots,m) and $\sum_{j=1}^{m} a_{ij} &lt; P_i$</td>
<td>No solutions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$P_i &gt; 0, a_{ij} &lt; P_i$ (j=1,2,\ldots,m) and $\sum_{j=1}^{m} a_{ij} = P_i$</td>
<td>All of appearing variables fixed</td>
<td>$x_1=x_2=\ldots=x_m=1$</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$P_i &gt; 0, a_{ij} &lt; P_i$ (j=1,2,\ldots,m)</td>
<td>One variable fixed</td>
<td>$x_1 = 1$</td>
<td>$\sum_{j=2}^{m} a_{ij} x_j &gt; P_i - a_{i1}$</td>
</tr>
<tr>
<td>6</td>
<td>$P_i &gt; 0, a_{ij} &gt; P_i$ (j=1,2,\ldots,m)</td>
<td>There are two possibilities</td>
<td>$\gamma_1: x_1=1$</td>
<td>$\sum_{j=2}^{m} a_{ij} x_j = P_i - a_{i1}$</td>
</tr>
</tbody>
</table>

Table 2.2 Inequality Constraints
Table 2.3 Preferential Order

<table>
<thead>
<tr>
<th>Preferential Order</th>
<th>Equation (Table 2.1)</th>
<th>Inequality (Table 2.2)</th>
<th>Characterization</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>1, 5</td>
<td>3</td>
<td>Determinate</td>
</tr>
<tr>
<td></td>
<td>2, 6</td>
<td>1, 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3, 7</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Second</td>
<td>4</td>
<td>2</td>
<td>Partially Determinate</td>
</tr>
<tr>
<td>Third</td>
<td>8</td>
<td>6</td>
<td>Indeterminate</td>
</tr>
</tbody>
</table>
each equation (or inequality) is subjected to the rules in a systematic manner. This fixes the value of some of the variables. The original system can be reduced to a much smaller system by substitution of these values. The repeated application of the rules ultimately results in the optimal solution.

Tables 2.1 and 2.2 represent three cases, namely when

(1) some of the variables are fixed;
(2) there is no solution; and
(3) the equation or inequality is redundant.

These cases may be referred to as determinate cases. In other cases there is no information available and the search has to be continued using the branching procedure. These can be called indeterminate cases. In yet other cases the search has to be extended to a number of possible values of the variables and can be called partially determinate cases. Table 2.3 shows this classification.

If some of the equations and inequalities are determinate, the available information is obtained and collated. Three situations may arise:

(1) an equation or inequality has no solution;
(2) two or more distinct equations or inequalities provide contradictory results; and
(3) the results are consistent.

In the first two cases, the system has no solution in the particular branch under investigation. The last case indicates a feasible solution and determines the values of certain variables.

If no determinate case exists in the system, the variable having the largest absolute value of coefficient in the objective function
serves as a node for branching. That is, it is assigned a value of 1 and 0 respectively and the resulting two branches are subjected to exploration.

To restrict the search to a smaller subset of all branches, a bounding procedure is utilized. An additional constraint is formed which places an upper bound on the objective function. The branches which indicate the value of the objective function in excess of this value are excluded from the search. Whenever a better result is obtained, it is utilized to improve the bounding.

An additional procedure, referred to as an accelerating process, is used to further limit the investigation to still fewer branches. Whenever a feasible solution is obtained in one branch, the technique will indicate whether a better solution exists or not in the second branch. In the latter case the search along the second branch is discontinued.

Thus, using the branching, bounding and accelerating processes, the complete enumeration of $2^n$ possible values is reduced considerably. By fixing the value of some of the variables, the properties of pseudo-Boolean functions further reduce the number of branches. The elimination process is, therefore, so devised that the optimal value always lies in the set in which the search is conducted. This guarantees the optimal value in finite number of steps.

2.2 Sample Problem

The linear pseudo-Boolean programming algorithm will be illustrated by the following sample problem.
Minimize
\[ z = 3x_1 + 6x_2 + x_6 \]

subject to
\[ x_1 + 5x_2 - 2x_3 - x_4 + x_5 - x_7 = 0 \]
\[ 4x_1 + 3x_2 - x_3 - 5x_4 - x_5 + x_6 - x_8 = 0 \]
\[ x_1 + x_3 = 1 \]
\[ x_2 + x_4 = 1 \]
\[ x_1 + x_2 = 1 \]

and
\[ x_j = 0 \text{ or } 1, \quad j = 1, 2, \ldots, 8 \]

The solution to the above problem is shown step by step as follows:

Step 1. Modify the problem. First, the problem is rewritten as minimize
\[ z = 3x_1 + 6x_2 + 0x_3 + 0x_4 + 0x_5 + 1x_6 + 0x_7 + 0x_8 \]

subject to
\[ 1x_1 + 5x_2 - 2x_3 - 1x_4 + 1x_5 + 0x_6 - 1x_7 + 0x_8 = 0 \]
\[ 4x_1 + 3x_2 - 1x_3 - 5x_4 - 1x_5 + 1x_6 - 0x_7 - 1x_8 = 0 \]
\[ 1x_1 + 0x_2 + 1x_3 + 0x_4 + 0x_5 + 0x_6 + 0x_7 + 0x_8 = 1 \]
\[ 0x_1 + 1x_2 + 0x_3 + 1x_4 + 0x_5 + 0x_6 + 0x_7 + 0x_8 = 1 \]
\[ 1x_1 + 1x_2 + 0x_3 + 0x_4 + 0x_5 + 0x_6 + 0x_7 + 0x_8 = 1 \]

and
\[ x_j = 0 \text{ or } 1, \quad j = 1, 2, \ldots, 8. \]

Second, a supplementary constraint is constructed such that
$$3x_1 + 6x_2 + 0x_3 + 0x_4 + 0x_5 + 1x_6 + 0x_7 + 0x_8 \leq 3 + 6 + 1,$$
$$3x_1 + 6x_2 + 0x_3 + 0x_4 + 0x_5 + 1x_6 + 0x_7 + 0x_8 \leq 10$$
or
$$-3x_1 - 6x_2 + 0x_3 + 0x_4 + 0x_5 - 1x_6 + 0x_7 + 0x_8 \geq -10.$$  
Finally, the negative sign in all the constraints is eliminated such that
$$-3(1-x_1) - 6(1-x_2) + 0x_3 + 0x_4 + 0x_5 - 1(1-x_6) + 0x_7 + 0x_8 \geq -10$$
$$1x_1 + 5x_2 - 2(1-x_3) - 1(1-x_4) + 1x_5 + 0x_6 - 1(1-x_7) + 0x_8 = 0$$
$$4x_1 + 3x_2 - 1(1-x_3) - 5(1-x_4) - 1(1-x_5) + 1x_6 + 0x_7 - 1(1-x_8) = 0$$
$$1x_1 + 0x_2 + 1x_3 + 0x_4 + 0x_5 + 0x_6 + 0x_7 + 0x_8 = 1$$
$$0x_1 + 1x_2 + 0x_3 + 1x_4 + 0x_5 + 0x_6 + 0x_7 + 0x_8 = 1$$
$$1x_1 + 1x_2 + 0x_3 + 0x_4 + 0x_5 + 0x_6 + 0x_7 + 0x_8 = 1$$
where
$$\bar{x}_j = 1 - x_j, \quad j = 1, 2, \ldots, 8$$
or
$$3\bar{x}_1 + 6\bar{x}_2 + 0\bar{x}_3 + 0\bar{x}_4 + 0\bar{x}_5 + 1\bar{x}_6 + 0\bar{x}_7 + 0\bar{x}_8 \geq 0$$
$$1x_1 + 5x_2 + 2\bar{x}_3 + 1\bar{x}_4 + 1x_5 + 0x_6 + 1\bar{x}_7 + 0x_8 = 4$$
$$4x_1 + 3x_2 + 1\bar{x}_3 + 5\bar{x}_4 + 1\bar{x}_5 + 1x_6 + 0x_7 + 1\bar{x}_8 = 8$$
$$1x_1 + 0x_2 + 1x_3 + 0x_4 + 0x_5 + 0x_6 + 0x_7 + 0x_8 = 1$$
$$0x_1 + 1x_2 + 0x_3 + 1x_4 + 0x_5 + 0x_6 + 0x_7 + 0x_8 = 1$$
$$1x_1 + 1x_2 + 0x_3 + 0x_4 + 0x_5 + 0x_6 + 0x_7 + 0x_8 = 1$$
Step 2. Branch from the term having the maximum coefficient in the supplementary constraint such that
\[ a_{1j}^* = \max_j [a_{1j}] = 6 \quad \text{and} \quad j = 2. \]
Substituting \( \bar{x}_2 = 1 \) in the system and simplifying, we get
\begin{align*}
3 & \bar{x}_1 + 1 \bar{x}_6 \geq -6 \tag{2.1} \\
1 & x_1 + 2 \bar{x}_3 + 1 \bar{x}_4 + 1 x_5 + 1 \bar{x}_7 = 4 \tag{2.2} \\
4 & x_1 + 1 \bar{x}_3 + 5 \bar{x}_4 + 1 x_5 + 1 x_6 + 1 \bar{x}_8 = 8 \tag{2.3} \\
1 & x_1 + 1 x_3 = 1 \tag{2.4} \\
x_4 &= 1 \tag{2.5} \\
x_1 &= 1. \tag{2.6}
\end{align*}
Step 3. Check for the determinate cases. Equation (2.5) gives
\[ x_4 = 1, \text{ (2.6) gives } x_1 = 1. \]
Substituting the values in all the constraints, we get
\begin{align*}
\bar{x}_6 & \geq -9 \tag{2.7} \\
2 & \bar{x}_3 + 1 x_5 + 1 \bar{x}_7 = 3 \tag{2.8} \\
1 & \bar{x}_3 + 1 \bar{x}_5 + 1 x_6 + 1 \bar{x}_8 = 4 \tag{2.9} \\
1 & x_3 = 0. \tag{2.10}
\end{align*}
Checking again for the determinate cases; equation (2.10)
gives \( \bar{x}_3 = 1 \) and equation (2.9) gives \( \bar{x}_5 = \bar{x}_6 = \bar{x}_8 = 1. \)
Substituting the values of \( \bar{x}_3 \) and \( x_5 \) in equation (2.8),
we get \( \bar{x}_7 = 1. \) All variables have been determined.

Step 4. Improve the bounding and apply the accelerating test. The feasible solution is given by
$x_1 = 1, \quad x_5 = 0,$
$x_2 = 0, \quad x_6 = 1,$
$x_3 = 0, \quad x_7 = 0,$
$x_4 = 1, \quad x_8 = 0.$

The value of the objective function is
\[
z = 3x_1 + 6x_2 + 1x_6
= 3(1) + 6(0) + 1(1)
= 4.
\]

Replace the supplementary constraint by
\[
3x_1 + 6x_2 + 1x_6 \leq 4,
- 3x_1 - 6x_2 - 1x_6 \geq - 4,
- 3(1-x_1) - 6(1-x_2) - 1(1-x_6) \geq - 4,
\]
or
\[
3\bar{x}_1 + 6\bar{x}_2 + \bar{x}_6 \geq 6.
\]

Now apply the accelerating test.

The coefficient of the branch point $\bar{x}_2 = a_{12} = 6$.

The variables $x_j$ in the branch which are having the value 1 if it is $\bar{x}_j$, or 0 if it is $x_j$ in the supplementary constraint are $x_1$ and $x_6$.

The sum of the coefficients of $x_1$ and $x_6$ is $3 + 1 = 4$.

Since $a_{12} > \text{sum of the coefficients (that is, 6 > 4)}$, the branch with $\bar{x}_2 = 0$ need not be investigated.

Thus the only feasible solution (and hence optimal) is given by
\[
x_1 = 1, \quad x_5 = 0,$
$x_2 = 0, \quad x_6 = 1,$
$x_3 = 0, \quad x_7 = 0,$
$x_4 = 1, \quad x_8 = 0.$
Minimum value of the objective function = 4.

Figure 2.1 shows the branching done in attaining the solution. The branch with $\bar{x}_2$ equal to 1 leads to a solution and the branch with $\bar{x}_2$ equal to 0 is terminated after applying the accelerating test. The value of the variables obtained in the branch is also shown.

Figure 2.1 Branching Tree for Sample Problem

2.3 Computational Algorithm

The linear pseudo-Boolean algorithm may be stated in a formal step by step procedure as follows:
Step 1. Modify the problem.

1.1 Set up the initial problem in the form

\[
\begin{align*}
\text{minimize} & \quad z = a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \\
\text{subject to} & \quad a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \geq \text{or } = P_2 \\
& \quad a_{31}x_1 + a_{32}x_2 + \ldots + a_{3n}x_n \geq \text{or } = P_3 \\
& \quad \vdots \\
& \quad a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \geq \text{or } = P_m
\end{align*}
\]

where

\[x_j = 0 \text{ or } 1, \quad j = 1, 2, \ldots, n,
\]

and

\[a_{ij} \text{'s and } P_i \text{'s are positive or negative integers.}
\]

1.2 Construct a supplementary constraint such that

\[
\begin{align*}
\text{minimize} & \quad z = a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \\
\text{subject to} & \quad a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq P_1
\end{align*}
\]

where \(P_1\) is the known upperbound on the objective function, or the sum of the positive coefficients.

Multiply next the supplementary constraint by \(-1\), which then takes the form,

\[
\begin{align*}
- a_{11}x_1 - a_{12} - \ldots - a_{1n}x_n \geq - P_1.
\end{align*}
\]

1.3 Eliminate the negative sign in all the constraints such that

\[
\begin{align*}
- a_{ij}x_j = - a_{ij}(1-x_j) \quad i = 1, 2, \ldots, m \\
& \quad j = 1, 2, \ldots, n,
\end{align*}
\]

where \(\bar{x}_j = (1-x_j)\) and \(a_{ij} \geq 0\)

Step 2. Branch from the term having the maximum coefficient in the supplementary constraint.
2.1 Select the term which has the maximum coefficient in the supplementary constraint,

\[ a_{j}^{*} = \max_{j} \{a_{ij}\} \]

If \( a_{j}^{*} = 0 \), check if \( a_{2j}^{*} = 0 \). Continue until a term \( a_{ij}^{*} \neq 0 \), \( i = 1, 2, \ldots, m \).

Substitute the value of \( x_{j} \) in the system such that

\[ x_{j} = \tilde{x}_{j} \]

where \( x_{j} = x_{j} \) or \( x_{j} \) appearing in the corresponding constraint and simplify the system.

2.2 No branch exists if

\[ a_{ij}^{*} = 0, \quad i = 1, 2, \ldots, m. \]

Then a feasible solution is obtained. Go to step 4.

Step 3. Check for the determinate cases.

3.1 If a determinate case exists, find the corresponding \( x \) values and substitute them in the system. Go to step 2.

3.2 If no determinate case exists, go to step 2 for further branching.

3.3 If infeasibility occurs, then there is no solution to the problem in that branch. Change the branch by setting \( \tilde{x}_{j} = 0 \).

3.4 If there is no feasible solution in either branch, return to the previous branch point and change the branch. Repeat this until (1) a feasible solution is obtained. Then go to step 2, or (2) all the branches are considered and no feasible solution exists; the search is then terminated.

Step 4. Improve the bounding and apply the accelerating test.
4.1 The feasible solution and the value of the objective function \( z \) are printed.

4.2 Replace the supplementary constraint by

\[
\begin{align*}
& a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq z, \\
& \text{Change the sign such that} \\
& a_{11}x_1 - a_{12}x_2 - \ldots - a_{1n}x_n \geq -z.
\end{align*}
\]

Eliminate the negative sign such that

\[
- a_{1j}x_j = - a_{1j}(1-x_j), \quad j = 1, 2, \ldots, n
\]

where

\[ x_j = (1-x_j) \text{ and } a_{1j} \text{'s are positive} \]

4.3 Apply the accelerating test.

Find the variables \( x_j \) in the branch \( x_j \) which are having the value \( x_j \) such that

\[
\begin{align*}
& x_j = \begin{cases} 
1, & \text{if } x_j = \bar{x}_j \\
0, & \text{if } \bar{x}_j = x_j 
\end{cases} \\
& \bar{x}_j = x_j
\end{align*}
\]

in the supplementary constraint.

Sum the coefficients of \( x_j \) in the objective function.

If \( a_{1j} \) is greater than the sum of the coefficients, the branch with \( x_j = 0 \) need not be investigated. Set \( J = J-1 \) and repeat the accelerating test.

If \( a_{1j} \) is less than or equal to the sum of the coefficients, the branch with \( x_j = 0 \) is investigated.

When \( J = 0 \), the final solution is optimal and the search is terminated.
CHAPTER III

APPLICATION TO COMBINATORIAL PROBLEMS

The combinatorial problem, as defined in Chapter I, is concerned with the study of the arrangement of elements into sets. In most industrial problems, the best one out of the possible arrangements has to be selected. Such problems are categorized as extremization problems. Shop scheduling, assembly-line balancing, delivery, travelling salesman, capital allocation and fixed-charge problems are different types of combinatorial problems. Since the solutions obtained must be integer-valued, these problems can be formulated as integer programming problems. By the proper utilization of zero-one variables, these problems can be converted into zero-one programming problems and can be solved by using a zero-one algorithm.

This chapter describes the formulations of shop scheduling, assembly-line balancing, delivery, travelling salesman, capital allocation and fixed-charge problem as zero-one programming problems. A sample problem in each type is presented and the associated solution discussed.

3.1 Shop Scheduling Problem

The shop scheduling problem in its simplest form consists of J jobs to be performed on M machines. Each job has a number of operations to be performed on the various machines in a prespecified machine ordering. It is required to determine a feasible sequence which results in the minimum completion time.

This problem can be formulated as a linear programming problem. The constraints which arise out of the inherent characteristics of the problem, are due to the following restrictions:
1. Each job is to be processed according to its machine ordering.
2. Each job should not be processed by more than one machine at the same time.
3. Two jobs should not be processed on the same machine simultaneously.
4. Each job has to be completed on a machine before the next job is performed on that machine.
5. In-process inventory is allowed.
6. The processing times are integer units and are known for all jobs.

The objective function and constraints are linear and therefore linear programming formulation provides a suitable approach. Since the results must be integers, integer linear programming is necessary. At present there exist three such formulations, due to Wagner [58], Bowman [11] and Manne [46]. Because of the smaller number of variables and constraints the formulation of the three-machine problem [25] is the only one that can be solved on computers due to the rapid increase in the number of constraints and variables.

The following notations are used in the formulation.

\[ J \quad \text{total number of jobs} \]

\[ j \quad \text{job designation, } j = 1, 2, \ldots, J \]

\[ j_k \quad \text{job } j \text{ in sequence position } k, \quad k = 1, 2, \ldots, J \]

\[ M \quad \text{total number of machines} = 3 \]

\[ m \quad \text{machine designation, } m = 1, 2, 3 \]

\[ t_{j_k}^m \quad \text{processing time of job } j_k \text{ on machine } m \]

\[ u_{j_k}^m \quad \text{waiting time of job in sequence position } k \text{ between machines } m \text{ and } m+1 \]
\[ v_{j_k^m} \] idle time on machine \( m \) between jobs in sequence position \\
\( k \) and \( k+1 \)

\[ x_{j_k^k} \] zero-one variable having a value one if job \( j \) is scheduled \\
in sequence position \( k \), zero otherwise

\[ x_{*k} \] a column vector \( [x_{1_k}, x_{2_k}, \ldots, x_{J_k}] \)

\[ p_{m} \] row vector of integer processing times for jobs \\
1, 2, \ldots, \( J \) on machine \( m \)

The three machines job shop problem is distinguished by the fact that, without loss of optimality, the search may be confined to schedules which sequence the \( J \) jobs in the same order on all three machines [55].

The constraints are given such that

1. A job \( j \) is assigned to the sequence position \( k \).
   \[ \sum_{j=1}^{J} x_{j_k} = 1, \quad k = 1, 2, \ldots, J \]

2. One of the sequence positions is assigned to job \( j \).
   \[ \sum_{k=1}^{J} x_{j_k} = 1, \quad j = 1, 2, \ldots, J \]

3. A job \( j \) is not processed on two machines simultaneously and a machine \( m \) does not process tow jobs at once.
   \[ v_{j_k^2} + p_{m} x_{j_k^{k+1}} + u_{j_k^{k+2}} - u_{j_k^2} - p_{m} x_{j_k^3} - v_{j_k^3} = 0 \]
   and
   \[ p_{m} x_{j_k^{k+1}} + u_{j_k^{k+2}} - u_{j_k^1} - p_{m} x_{j_k^2} - v_{j_k^2} = 0 \]
   \[ k = 1, 2, \ldots, (J-1) \]
It has been shown by Johnson [39] and Bellman [9] that minimizing the total time span to complete all items is equivalent to minimizing the idle time on machine 3. Hence Wagner's formulation suggests the following form of objective function.

Minimize

\[ Z = [P_2 + P_3] X_1 + \sum_{k=1}^{J-1} v_{j_k}^3. \]

In such a formulation, the total number of variables is \(J^2 + 4(J-1)\) and the number of constraints becomes \((4J-3)\). The integer valued variables \(u_{j_k}^3\) and \(v_{j_k}^3\) are converted into zero-one variables using Balas binary technique.

The following sample problem will illustrate the above formulation.

Consider a flow shop problem having the following machine ordering and processing time matrix. It is required to minimize the total processing time.

\[
\begin{bmatrix}
11 & 12 & 13 \\
21 & 22 & 23
\end{bmatrix}
\quad \quad
\begin{bmatrix}
2 \\
1 
\end{bmatrix}
\quad \quad
\begin{bmatrix}
1 & 4 \\
5 & 3
\end{bmatrix}
\]

The objective function is to minimize

\[ f = \left( P_1 + P_2 \right) X_1 + \sum_{i=1}^{J-1} v_{j_i}^3 \]

\[ = \left( \frac{2}{1} \right) X_1^1 + \left( \frac{1}{5} \right) X_1^2 + v_{j_1}^3 \]

\[ = 3X_1^1 + 6X_2^1 + v_{j_1}^3. \]
The constraints are given such that

1. One of the job \( j \) is assigned to the sequence position \( k \).

\[
\sum_{j=1}^{J} x_{j_k} = 1 \quad k = 1, 2
\]

\[
x_{1_1} + x_{2_1} = 1
\]

and \( x_{1_2} + x_{2_2} = 1 \)

2. One of the sequence position is assigned to a job \( j \).

\[
\sum_{k=1}^{J} x_{j_k} = 1 \quad j = 1, 2
\]

\[
x_{1_1} + x_{1_2} = 1
\]

and \( x_{1_1} + x_{2_2} = 1 \)

In the above four equations, one equation is redundant and therefore can be dropped.

3. A job \( j \) is not processed on two machines simultaneously and a machine \( m \) does not process two jobs at once.

\[
(4, 3) \begin{bmatrix} x_{1_1} \\ x_{2_1} \end{bmatrix} = (1, 5) \begin{bmatrix} x_{1_2} \\ x_{2_2} \end{bmatrix} + v_{j_1 2} + v_{j_1 3} - u_{j_2 2} = 0
\]

\[
4x_{1_1} + 3x_{2_1} - x_{1_2} - 5x_{2_2} - v_{j_1 2} + v_{j_1 3} - u_{j_2 2} = 0
\]

and

\[
(2, 1) x_{1_1} - (1, 2) x_{2_1} + v_{j_1 2} - u_{j_2 1} = 0
\]
\[
\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{pmatrix} - \begin{pmatrix} 2 \\ 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + v_{j_12} - u_{j_21} = 0
\]

\[x_1 + 5x_2 - 2x_3 - x_4 + v_{j_12} - u_{j_21} = 0\]

Substituting zero-one variables \(x_j, j = 1, 2, \ldots, 8\) as shown below,

\[
x_1 = x_{1_1}, \quad x_5 = v_{j_12},
\]
\[
x_2 = x_{2_1}, \quad x_6 = v_{j_13},
\]
\[
x_3 = x_{1_2}, \quad x_7 = u_{j_21},
\]
\[
x_4 = x_{2_2}, \quad x_8 = u_{j_22}
\]

the problem reduces to the following:

minimize

\[
f = 3x_1 + 6x_2 + x_6
\]

subject to

\[4x_1 + 3x_2 - x_3 - 5x_4 - x_5 + x_6 - x_8 = 0\]
\[x_1 + 5x_2 - 2x_3 - x_4 - x_5 - x_7 = 0\]
\[x_1 + x_3 = 1\]
\[x_2 + x_4 = 1\]
\[x_1 + x_2 = 1\]

and

\[x_j = 0 \text{ or } 1, \quad j = 1, 2, \ldots, 8.\]

The total number of zero-one variables is 8 and the number of constraints is 5. The solution of this problem is demonstrated in Section 2.2. The solution is given by
\[ x_1 = 1, \quad x_5 = 0 \]
\[ x_2 = 0, \quad x_6 = 1 \]
\[ x_3 = 0, \quad x_7 = 0 \]
\[ x_4 = 1, \quad x_8 = 0 \]

and

\[ \text{minimum } f = f^* = 4. \]

Minimum schedule time is given by the following:

\[ \text{minimum schedule time} = \text{processing time of 2 jobs on machine 3} + f^* \]
\[ = 4 + 3 + 4 \]
\[ = 11. \]

\[ x_1 = x_{11} = 1 \] and \[ x_4 = x_{22} = 1 \] indicate that the optimal sequence
\[ S^* = \{1, 2\}. \]

The optimal schedule is represented on the Gantt chart in Figure 3.1.

![Gantt Chart](image)

**Figure 3.1** Gantt Chart for a (2x3) Flow-Shop Sample Problem
3.2 Assembly-line Balancing Problem

An assembly-line consists of a number of work stations. To assemble a product, a number of tasks must be performed subject to certain sequencing requirements concerning the order in which they are performed. Given a cycle time, the assembly-line balancing problem consists of minimizing the number of work stations.

The following notations are used in the formulation due to Bowman [12].

\[ K \] total number of work stations
\[ J \] total number of tasks
\[ X_j \] the initial time when task \( j \) is started \( j = 1, 2, \ldots, J \)
\[ I_{cd} \] \( \begin{cases} 1, & \text{if task } c \text{ precedes task } d \\ 0, & \text{otherwise} \end{cases} \]
\[ T \] maximum clock time a product takes to come out of the assembly-line
\[ c \] cycle time
\[ t_j \] processing time for task \( j \), \( j = 1, 2, \ldots, J \)
\[ \tau \] number of time units the product is on the assembly-line
\[ u_j \] integer-valued variable which can take any value from 0 to \( X_j \), \( j = 1, 2, \ldots, J \)

The constraints are given such that

1. Each task is performed in accordance with the ordering requirements.
   \[ X_j + t_j \leq X_{j+1}, \quad j = 1, 2, \ldots, J-1. \]
2. Each work station can take up a task only after it leaves the previous station.
\[(T + t_{j+1}) I_j^{(j+1)} + (X_j - X_{j+1}) \geq t_{j+1}\]
and
\[(T + t_j) (1 - I_j^{(j+1)}) + (X_{j+1} - X_j) \geq t_j, \quad j = 1, 2, \ldots, J-1.\]

3. Each work station should not be overloaded and the tasks must be completed before being passed on to the next station.

\[X_j + t_j \leq cu_j + c\]

and
\[X_j \geq cu_j, \quad j = 1, 2, \ldots, J.\]

4. All operations are over within the total completion time with no followers in a specified ordering.

\[X_S + t_S \leq \tau \quad \text{for each } S,\]

where

\[S \text{ is a set of stations without any succeeding stations.}\]

The objective of minimizing the number of work stations is to distribute the work load uniformly on all work stations. This will reduce the number of time units the product is on the assembly-line. Hence the objective function becomes minimize

\[z = \tau.\]

The formulation utilizes 2J+1 integer-valued variables and about J zero-one variables (the exact number depends on the ordering requirements). The total number of constraints is about 5J (again the exact number depends on the ordering requirements).

The following sample problem illustrates the formulation of the assembly-line balancing problem as an integer programming problem.
Consider an assembly-line as shown in Figure 3.2. The ordering and the initial times are shown for the four tasks as shown below. It is required to reduce the number of time units the product is on the assembly-line.

![Diagram of tasks and times]

**Fig. 3.2 Ordering Position for Sample Problem.**

<table>
<thead>
<tr>
<th>Station</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial time</td>
<td>1-20</td>
<td>21-40</td>
<td>41-60</td>
<td>61-80</td>
</tr>
</tbody>
</table>

The objective is to minimize the total number of time units the product is on the assembly line. Hence the objective function is given by minimize

\[ z = \tau. \]

The constraints are such that

1. Each task is done in accordance with the ordering requirements.

\[
X_j + t_j \leq X_{j+1}, \quad j = 1, 2, \ldots, J-1
\]
\[
X_a + 11 \leq X_b
\]
\[
X_b + 14 \leq X_c
\]
\[
X_c + 14 \leq X_d
\]
2. Each workstation can take up a task only after it leaves the
previous station.

\[(T + t_{j+1}^\prime) I_{j(j+1)} + (x_j - x_{j+1})\geq t_{j+1}\]

and

\[(T + t_j^\prime)(1 - I_{j(j+1)}) + (x_{j+1} - x_j)\geq t_j, \quad j = 1, 2, \ldots, J-1\]

\[(80 + 1) I_{ab} + (x_a - x_b)\geq 14\]
\[(80 + 11)(1 - I_{ab}) + (x_b - x_a)\geq 11\]
\[(80 + 9) I_{bc} + (x_c - x_c)\geq 9\]
\[(80 + 14)(1 - I_{bc}) + (x_c - x_b)\geq 14\]
\[(80 + 5) I_{bd} + (x_b - x_d)\geq 5\]
\[(80 + 14)(1 - I_{bd}) + (x_d - x_b)\geq 14\]

3. Each workstation is not overloaded and the tasks must be completed
before being passed on to the next station.

\[x_j + t_j \leq c u_j + c\]

and

\[x_j \geq c u_j, \quad j = 1, 2, \ldots, J\]
\[x_a + 11 \leq 20 u_a + 20\]
\[x_a \geq 20 u_a\]
\[x_b + 14 \leq 20 u_b + 20\]
\[x_b \geq 20 u_b\]
\[x_c + 9 \leq 20 u_c + 20\]
\[x_c \geq 20 u_c\]
\[x_d + 5 \leq 20 u_d + 20\]
\[x_d \geq 20 u_d\]

4. All operations are completed within the total completion time with
no followers in a required ordering.
\[ X_s + t_s \leq \tau \]
\[ X_c + 9 \leq \tau \]
\[ X_d + 5 \leq \tau \]

This problem utilizes 9 integer-valued variables and 3 zero-one variables. The total number of constraints is 19. The integer-valued variables are converted to zero-one variables using Balas binary technique in which 7 zero-one variables are used for each of the integer-valued variables \( X_a \) to \( X_d \), 3 zero-one variables are used for each of the integer-valued variables \( u_a \) to \( u_d \) and 7 zero-one variables are used for \( \tau \). This substitution results in the problem size of 50 variables and 19 constraints.

Solving this problem by zero-one programming, we get

\[
\begin{align*}
X_a &= 0, & u_a &= 0 \\
X_b &= 23, & u_b &= 1 \\
X_c &= 40, & u_c &= 1 \\
X_d &= 43, & u_d &= 1 \\
I_{ab} &= I_{bc} = I_{bd} &= 1
\end{align*}
\]

and

minimum \( \tau = 49 \)

This is the minimum time that a job takes to come out of the assembly-line. Stations 'b' and 'd' are grouped together. The job takes 20 units of cycle time in Stations 'a' and 'b'. After completing 9 time units in Station 'c' the job emerges from the assembly-line, thus requiring a total of 49 time units.

3.3 Delivery Problem

The delivery problem arises whenever commodities are to be transported from a central warehouse to a number of customers at different
destinations within a specified region. The orders received at the warehouse are grouped and delivered in batches. The deliveries are arranged so that each customer receives his entire order in one delivery but the delivery schedules are set by the shipper on the basis of the availability of carriers. The objective of the shipper is to minimize the total cost of transportation in fulfilling customer orders.

The following notations are used in the formulation due to Balinski and Quandt [5].

m number of destinations
n number of feasible combination of orders - number of activities
A_j activities column vector each having m entities. The i^{th} entry of A_j = 1, if activity j delivers order i and A_j = 0 otherwise
j = 1, 2, ..., n
c_j cost of the activity A_j
r number of possible geographical routes
E column vector of m 1's
X_j zero-one variable having a value 1 if the activity A_j is used, zero otherwise.

The constraints are given such that

1. A given carrier can combine a number of orders to be delivered together, provided their destinations lie along one of a number of permissible geographical routes and a given destination can receive delivery via a number of different routes.

\[ \sum_{j=1}^{n} A_j x_j = E \]
The objective is to minimize total shipping cost.

Minimize

\[ z = \sum_{j=1}^{n} c_j x_j \]

The total number of zero-one variables used in this formulation is \( n \) and the total number of constraints becomes \( m \).

The following sample problem will illustrate the formulation of the delivery problem as a zero-one integer programming problem.

Consider a warehouse shipping orders to 4 destinations as shown in figure 3.2. The total number of permissible geographical routes, \( m = 4 \). The number of activities \( n \) and associated costs are as shown below. The objective is to minimize total cost of transportation.

![Diagram of delivery routes](image)

**Fig. 3.3 Delivery Routes for Sample Problem.**

\[
\begin{align*}
A_1 &= \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} ; & A_2 &= \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} ; & A_3 &= \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} ; & A_4 &= \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \\
& & c_1 &= 6 ; & c_2 &= 8 ; & c_3 &= 9 ; & c_4 &= 6
\end{align*}
\]
The delivery problem can now be formulated as

minimize

\[ z = \sum_{j=1}^{4} c_j x_j \]

\[ = 6x_1 + 8x_2 + 9x_3 + 6x_4 \]

subject to

\[ \sum_{j=1}^{n} A_j x_j = E \]

\[ \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} x_1 + \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} x_2 + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} x_3 + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} x_4 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \]

or

\[ x_1 + x_2 = 1 \]

\[ x_3 + x_4 = 1 \]

\[ x_1 + x_3 = 1 \]

\[ x_2 + x_4 = 1 \]

and

\[ x_j = 0 \text{ or } 1, \quad j = 1, 2, 3, 4. \]

The total number of zero-one variables is 4 and the number of constraints is 4. Solving this problem by zero-one programming, we get

\[ x_1 = x_4 = 1, \]

\[ x_2 = x_3 = 0 \]

and

\[ \text{minimum } z = 12. \]

This indicates that shipping in the routes 1 and 4 will minimize the transportation cost.
3.4 Travelling Salesman Problem

The travelling salesman problem in simple terms may be stated as follows. A salesman, starting from one city, visits each of the other \( n \) cities once and only once and returns to the starting city. The problem is to find the order in which he should visit the cities to minimize the total distance traveled. Any other measure of effectiveness such as time or cost may be substituted for distance. This measure of effectiveness between all pairs of cities are presumed to be known.

The distances between the city pairs can be arranged in a matrix form. Since it is not possible to travel from one city to the same city in one step, the corresponding element in the matrix is a very large value. Thus an infinitely large number is placed in each element on the diagonal of such a matrix.

The following notations are used in the formulation due to Miller et al. [48].

\[
\begin{align*}
    n & \quad \text{number of cities to be visited} \\
    d_{ij} & \quad \text{distance from city } i \text{ to city } j, \quad i = 0, 1, 2, \ldots, n \\
    j = 0, 1, 2, \ldots, n \\
    u_i & \quad \text{arbitrary real-valued variables used to eliminate subtours} \quad i = 1, 2, \ldots, n \\
    x_{ij} & \quad \text{zero-one variable having a value of one if the salesman proceeds from city } i \text{ to city } j, \text{ zero otherwise}
\end{align*}
\]

The constraints are given such that

1. Arrival at each city from any other city is only once excluding the starting city which can be visited any number of times.
\[ \sum_{i=0}^{n} x_{i,j} = 1, \quad j = 1, 2, \ldots, n. \]
\[ \sum_{j=0}^{n} x_{i,j} = 1, \quad i = 1, 2, \ldots, n. \]

2. Departure from each city to any other city is once only excluding the starting city which can be visited any number of times.

\[ \sum_{j=0}^{n} x_{i,j} = 1, \quad i = 1, 2, \ldots, n. \]
\[ j \neq i \]

3. Tour should commence and end at the starting city and no tour should visit more than \( n \) cities.

\[ u_i - u_j + n x_{i,j} \leq n - 1, \quad 1 \leq i \neq j \leq n. \]

The objective is to minimize the total distance covered and hence the objective is given by

minimize

\[ z = \sum_{0<i,j<n} \sum_{0<i,j<n} d_{i,j} x_{i,j}. \]

The total number of variables is \( n^2 + 2n \) and the number of constraints becomes \( n^2 + n \). The integral-valued variables \( u_i (i=1, 2, \ldots, j) \) are converted into zero-one variables using Balas binary technique.

The following sample problem will illustrate the integer linear programming formulation of the travelling salesman problem. Consider a problem in which there are 3 cities to be visited starting from city 0. The distance matrix is as shown below and it is required to find the route which minimizes the total distance travelled.

\[
D = \begin{pmatrix}
0 & 1 & 2 & 3 \\
0 & \infty & 2 & 3 \\
1 & 5 & \infty & 2 & 6 \\
2 & 3 & 5 & \infty & 4 \\
3 & 4 & 3 & 5 & \infty
\end{pmatrix}
\]
The objective function is to minimize the total distance.

Minimize

\[ z = \sum_{0 \leq i < j \leq 1} \sum d_{ij} x_{ij} \]

\[ = d_{01} x_{01} + d_{02} x_{02} + d_{03} x_{03} + d_{10} x_{10} + d_{12} x_{12} + d_{13} x_{13} \]

\[ + d_{20} x_{20} + d_{21} x_{21} + d_{23} x_{23} + d_{30} x_{30} + d_{31} x_{31} + d_{32} x_{32} \]

\[ = 4x_{01} + 2x_{02} + 3x_{03} + 5x_{10} + 2x_{12} + 6x_{13} \]

\[ + 3x_{20} + 5x_{21} + 4x_{23} + 4x_{30} + 3x_{31} + 5x_{32} \]

The constraints are given such that

1. Arrival to each city from any other city is only once.

\[ \sum_{i=0}^{n} \sum_{i \neq j} x_{ij} = 1 \quad j = 1, 2, \ldots, n. \]

\[ x_{01} + x_{21} + x_{31} = 1 \]

\[ x_{02} + x_{12} + x_{32} = 1 \]

\[ x_{03} + x_{13} + x_{23} = 1 \]

2. Departure from each city to any other city is only once.

\[ \sum_{j=0}^{n} \sum_{j \neq i} x_{ij} = 1 \quad i = 1, 2, \ldots, n. \]

\[ x_{10} + x_{12} + x_{13} = 1 \]

\[ x_{20} + x_{21} + x_{23} = 1 \]

\[ x_{30} + x_{31} + x_{32} = 1 \]

3. Tour should commence and end at the starting city, and no tour should cover more than n cities.
\[ u_i - u_j + nX_{ij} \leq n - 1 \quad 1 \leq i \neq j \leq n \]
\[ u_1 - u_2 - 3X_{12} \leq 2 \]
\[ u_1 - u_3 - 3X_{13} \leq 2 \]
\[ u_2 - u_1 - 3X_{21} \leq 2 \]
\[ u_2 - u_3 - 3X_{23} \leq 2 \]
\[ u_3 - u_1 - 3X_{31} \leq 2 \]
\[ u_3 - u_2 - 3X_{32} \leq 2 \]

The three-city problem results in a 15 variables, 9 constraints integer linear programming problem. The following substitution is made to convert the problem into zero-one integer programming problem.

\[
x_1 = X_{01}, \quad x_7 = X_{20}
\]
\[
x_2 = X_{02}, \quad x_8 = X_{21}
\]
\[
x_3 = X_{03}, \quad x_9 = X_{23}
\]
\[
x_4 = X_{10}, \quad x_{10} = X_{30}
\]
\[
x_5 = X_{12}, \quad x_{11} = X_{31}
\]
\[
x_6 = X_{13}, \quad x_{12} = X_{32}
\]
\[
8x_{13} + 4x_{14} + 2x_{15} + x_{16} = u_1
\]
\[
8x_{17} + 4x_{18} + 2x_{19} + x_{20} = u_2
\]
\[
8x_{21} + 4x_{22} + 2x_{23} + x_{24} = u_3
\]

The problem now reduces to minimize

\[
z = 4x_1 + 2x_2 + 3x_3 + 5x_4 + 2x_5 + 6x_6
\]
\[
+3x_7 + 5x_8 + 4x_9 + 4x_{10} + 3x_{11} + 5x_{12}
\]

subject to
\[ x_1 + x_8 + x_{11} = 1 \]
\[ x_2 + x_5 + x_{12} = 1 \]
\[ x_3 + x_6 + x_9 = 1 \]
\[ x_4 + x_5 + x_6 = 1 \]
\[ x_7 + x_8 + x_9 = 1 \]
\[ x_{10} + x_{11} + x_{12} = 1 \]

\[ 8x_{13} + 4x_{14} + 2x_{15} + x_{16} - 8x_{17} - 4x_{18} - 2x_{19} - x_{20} - 3x_5 \leq 2 \]
\[ 8x_{13} + 4x_{14} + 2x_{15} + x_{16} - 8x_{21} - 4x_{22} - 2x_{23} - x_{24} - 3x_6 \leq 2 \]
\[ 8x_{17} + 4x_{18} + 2x_{19} - x_{20} - 8x_{13} - 4x_{14} - 2x_{15} - x_{16} - 3x_8 \leq 2 \]
\[ 8x_{17} + 4x_{18} + 2x_{19} + x_{20} - 8x_{21} - 4x_{22} - 2x_{23} - x_{24} - 3x_9 \leq 2 \]
\[ 8x_{21} + 4x_{22} + 2x_{23} + x_{24} - 8x_{13} - 4x_{14} - 2x_{15} - x_{16} - 3x_{11} \leq 2 \]
\[ 8x_{21} + 4x_{22} + 2x_{23} + x_{24} - 8x_{17} - 4x_{18} - 2x_{19} - x_{20} - 3x_{12} \leq 2 \]

and

\[ x_j = 0 \text{ or } 1, \quad j = 1, 2, \ldots, 24. \]

The total number of zero-one variables is 24 and the number of constraints is 5. The solution of the problem is given by

\[ x_{13} = 1, \quad x_{10} = 1 \]
\[ x_{14} = 1, \quad x_{11} = 1 \]

and

minimum \( z = 11. \)

This indicates that the salesman travels from city 0 to 3, 3 to 1, 1 to 2 and 2 to 0 resulting in a minimum distance of 11 units.

3.5 Capital Allocation Problem

The allocation problem arises in the capital budgeting of a firm. It consists of finding an optimal way in which a firm should allocate the available capital to various projects. This problem can be formulated as an integer programming problem due to Weingartner [33].
The following notations are used in the formulation.

\( n \)  
total number of projects under consideration

\( b \)  
total amount of investment available

\( c_j \)  
present worth of all future profits from project \( j \), \( j = 1, 2, \ldots, n \)

\( d_j \)  
amount of capital required for project \( j \), \( j = 1, 2, \ldots, n \)

\( x_j \)  
zero-one variable having a value one if project \( j \) is taken, zero otherwise

The constraint is such that

1. The total capital invested on all the projects undertaken is less than or equal to the capital available.

\[
\sum_{j=1}^{n} d_j x_j \leq b
\]

The objective is to maximize the present worth of all the future profits from the projects undertaken and is given by

\[
\text{maximize } z = \sum_{n=1}^{n} c_j x_j .
\]

The total number of zero-one variables is \( n \) and the constraint is one only.

The following sample problem will illustrate the above formulation.

Consider a case where there are 10 projects under consideration. The total available capital is 55. The amount of capital required for the projects and the present worth of all future profits from the projects is as shown below.

\( d_1 = 30; \quad c_1 = 20 \)

\( d_2 = 25; \quad c_2 = 18 \)
\[ d_3 = 20; \quad c_3 = 17 \]
\[ d_4 = 18; \quad c_4 = 15 \]
\[ d_5 = 17; \quad c_5 = 15 \]
\[ d_6 = 11; \quad c_6 = 10 \]
\[ d_7 = 5; \quad c_7 = 5 \]
\[ d_8 = 2; \quad c_8 = 3 \]
\[ d_9 = 1; \quad c_9 = 1 \]
\[ d_{10} = 1; \quad c_{10} = 1 \]

The objective is to select the projects such that the present worth of all future profits is maximized.

The problem can now be formulated as

\[
\text{maximize} \quad z = \sum_{j=1}^{10} c_j x_j \\
= 20x_1 + 18x_2 + 17x_3 + 15x_4 + 15x_5 + 10x_6 + 5x_7 + 3x_8 + x_9 + x_{10}
\]

subject to

\[
\sum_{j=1}^{n} d_j x_j \leq b
\]

or

\[
30x_1 + 25x_2 + 20x_3 + 18x_4 + 17x_5 + 11x_6 + 5x_7 + 2x_8 + x_9 + x_{10} \leq 55
\]

and

\[ x_j = 0 \text{ or } 1, \quad j = 1, 2, \ldots, 10. \]

Solving this problem of 10 variables and 1 constraint the solution yields

\[ x_1 = x_2 = x_3 = 0, \]
\[ x_4 = x_5 = x_6 = x_7 = x_8 = x_9 = x_{10} = 1 \]
and a maximum profit of 50.

This indicates that the available of 55 units is distributed to projects 4, 5, 6, 7, 8, 9 and 10. The projects 1, 2 and 3 are dropped. This decision results in a maximum profit of 50 units.

3.6 Fixed-Charge Problem

The fixed-charge problem arises in situations where a certain fixed amount of cost is incurred whenever an activity takes place. The corresponding costs are known as fixed-charges. For example, in transportation, a fixed-charge is incurred regardless of the quantity shipped, or in the building of production facilities where a plant under construction must have a certain minimum size. Because of these fixed-charges, such problems attain special characteristics. If there is a fixed-charge associated with each variable, then every extreme point of the convex set of feasible solutions yields a local optimum and this complicates the task of solving fixed-charge problems.

The following notations are used in the formulation due to Hadley [33].

\begin{align*}
  & n \quad \text{number of activities} \\
  & f_j \quad \text{fixed-charge for activity } X_j, \quad j = 1, 2, \ldots, n \\
  & c_j \quad \text{variable cost of activity } j, \quad j = 1, 2, \ldots, n \\
  & A \quad \text{coefficient matrix} \\
  & P \quad \text{column vector of right hand side}
\end{align*}
\( d_j \) zero-one variable having a value 1 if the activity \( X_j \) is used, zero otherwise, \( j = 1, 2, \ldots, n \)

\( u_j \) upper bound on the variable \( X_j \), \( j = 1, 2, \ldots, n \)

\( x \) column vector of \( X_j \), \( j = 1, 2, \ldots, n \)

The constraints are given such that

1. The sum of the resources needed for all activities is equal to the available resources.
   \[
   A x = P
   \]

2. A fixed-charge is incurred when an activity \( x_j \) is used
   \[
   x_j - u_j d_j \leq 0, \quad j = 1, 2, \ldots, n
   \]

The objective is to minimize the total cost incurred and is given by

\[
\text{minimize} \quad z = \sum_{j=1}^{n} (f_j x_j + c_j x_j).
\]

The total number of integral-valued variables \( X_j \) is \( n \). This is converted to zero-one variable using Balas binary technique.

The following sample problem will illustrate the above formulation.

Consider a case where there are 3 activities each with a fixed-charge of 1 and variable cost of 1. The upper bounds on \( X_1, X_2, \) and
$X_3$ are given by 5, 4 and 3 respectively. The problem is to minimize
\[ z = 2X_1 + 2X_2 + 2X_3 \]
subject to
\[ X_1 + X_2 + X_3 = 6 \]
\[ 2X_1 + X_2 + 3X_3 = 10 \]
\[ X_1 - 5d_1 \leq 0 \]
\[ X_2 - 4d_2 \leq 0 \]
\[ X_3 - 3d_3 \leq 0 \]
and
\[ x_j \geq 0 \quad j = 1, 2, \ldots, n. \]
The following substitution is made to convert the problem into a zero-one integer programming problem.
\[
\begin{align*}
4x_1 + 2x_2 + x_3 &= x_1 \\
4x_4 + 2x_5 + x_6 &= x_2 \\
2x_7 + x_8 &= x_3 \\
x_9 &= d_1 \\
x_{10} &= d_2 \\
x_{11} &= d_3 
\end{align*}
\]
The problem now reduces to the following:
minimize
\[ z = 8x_1 + 4x_2 + 2x_3 + 8x_4 + 4x_5 + 2x_6 + 4x_7 + 2x_8 \]
subject to
\[
\begin{align*}
4x_1 + 2x_2 + x_3 + 4x_4 + 2x_5 + x_6 + 2x_7 + x_8 &= 6 \\
8x_1 + 4x_2 + 2x_3 + 4x_4 + 2x_5 + x_6 + 6x_7 + 3x_8 &= 10 \\
4x_1 + 2x_2 + x_3 - 5x_9 &\leq 0
\end{align*}
\]
\[ 4x_4 + 2x_5 + x_6 - 4x_{10} \leq 0 \]
\[ 2x_7 + x_8 - 3x_{11} \leq 0 \]

and

\[ x_j = 0 \text{ or } 1, \quad j = 1, 2, \ldots, 11. \]

The total number of zero-one variables is 11 and the number of constraints is 5. Solving this problem by zero-one programming, we get

\[ x_1 = 4, \]
\[ x_2 = 2, \]
\[ x_3 = 0 \]

and the value of the objective function

\[ z = 12. \]

This indicates that activity 1 and 2 are used and activity 3 is dropped with the resultant minimum cost of 12.
CHAPTER IV

COMPUTATIONAL EXPERIENCE

This chapter comprises of two sections. The first section includes the experience obtained in solving various combinatorial problems, namely, shop scheduling, assembly-line balancing, delivery, traveling salesman, capital allocation and fixed-charge problems were solved on IBM 360/50 by using the pseudo-Boolean program. The same problems were solved using DZLP developed by Salkin and Spielberg [54]. The computational time taken by the two programs are compared and discussed in Section 4.1. The computational difficulties faced in solving the problems are discussed in Section 4.2.

4.1 Results of the Pseudo-Boolean Algorithm

Results using the pseudo-Boolean algorithm discussed in Chapter II are detailed in this section. Capital allocation [57] and fixed-charge problems [34] were taken from the literature and all other problems were randomly generated. The problems were converted to zero-one programming problem as indicated in Chapter III. Each equality constraint had to be broken into two inequality constraints when DZLP was used. Since a pseudo-Boolean program could handle equality constraints, they were retained.

The flow shop problems have all equality constraints. The constraints arise mainly due to sequencing and non interference restrictions. The (3x3) problem requires about 33 zero-one variables and 9 constraints whereas a problem of size (4x3) utilizes 52 zero-one variables and
13 constraints. The variables increase quadratically with the increase in the number of jobs but the increase in the number of constraints is only linear. Since the total number of branches to be investigated is $2^n$ for n variables, the computation time increases non-linearly with the increase in the number of jobs. The constraints include an "assignment constraint matrix" and this favours the computational aspect of the pseudo-Boolean programming by fixing the values of many variables in one branch. This process reduces the number of branches to be investigated to a great extent. Two problems in (4x3) flow shop problems did not converge within 15 minutes and the problem was terminated while using the pseudo-Boolean programming. This was due to the large number of branches generated in these problems and the fixation of values of the variables in the branches was poor.

The assembly-line balancing problems have all inequality constraints. A large number of matrix and cost coefficients are zero. The constraints arise mainly due to ordering and non-interference restrictions. A 4-task problem requires about 50 zero-one variables and 19 constraints and an 8-task problem about 96 zero-one variables and 42 constraints. The increase in the number of variables and constraints is linear. Because of the absence of "assignment matrix constraints", the fixation of values to the variables in various branches was very poor. This increases the number of branches and the amount of search to a great extent. The convergence was very slow while using the pseudo-Boolean programming and the program had to be terminated after 15 minutes without reaching the optimal value.
The delivery problems have all equality constraints with the
constraint matrix coefficients being either zero or one. The con-
straints arise mainly to satisfy the requirement that the demand
is equal to the supply and the commodity has to be shipped in one
of the permissible geographical routes. The total number of zero-one
variables is equal to the number of feasible combinations of orders
and the number of constraints is equal to the number of destinations.
The increase in the number of variables and the number of constraints
is linear. Due to the zero-one coefficients and unit right hand
side in the constraint matrix the number of branches is reduced
to a great extent and the search converges very rapidly. In case of
DZLP the equality constraints were split into two inequality con-
straints. The greater than or equal to constraints were converted
to less than or equal to constraints. The DZLP fixed all the variables
at zero value thereby violating the constraints. It failed to reach
the optimal value. Several parameter modifications were tried without
any success. The reason for this failure could not be determined.

The traveling salesman problems have half equality and half in-
equality constraints. The constraints arise due to the fact that
each city should be visited only once without any overlapping of the
tour. The increase in the number of variables and constraints is
quadratic with the increase in the number of cities to be visited.
This fact imposes a severe restriction on the size of the problem
that can be solved by utilizing this formulation. The constraints in-
clude an "assignment constraint matrix" and this favours pseudo-Boolean
programming. The convergence was very good while using the pseudo-Boolean
programming.
All the nine capital allocation problems are the same except for the right hand side of the constraint matrix. These problems differ from others by having dense coefficient matrix. That is, all the coefficients are greater than zero. Pseudo-Boolean programming shows better results in solving the capital allocation problems.

The solution of the fixed-charge problems is made difficult by a number of local optimal solutions which obscure the global optimum. Results on nine test problems from Halden [34] indicate that the convergence of pseudo-Boolean programming is faster than that of DZLP in solving fixed-charge problems.

4.2 Computational Difficulties

The size of the problem which can be solved by using the pseudo-Boolean algorithm has to be restricted because of the large storage requirements. An attempt was made to use H level Fortran but it had to be discontinued since the H level program was not running smoothly.

It was observed that a considerable amount of time is spent in substituting the value of the variable obtained in one equation or inequality, in all the remaining equations and/or inequalities and simplifying the system. Further, if one variable is fixed in a simplification, again the substitution and simplification are to be made which consume a lot of computer time. Several different methods were tried to reduce this time. It was found that starting the constraints with equations, if any, produced better results.

An 8-task line balancing problem taken from Bowman [12] was tried in DZLP. It resulted in a problem size of 96 variables and 42 constraints. The program failed to attain the optimum value within one hour and has to be terminated.
<table>
<thead>
<tr>
<th>Problem No.</th>
<th>Problem Size (JxM)</th>
<th>pseudo-Boolean Program</th>
<th></th>
<th>DZLP</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No. of Variables</td>
<td>No. of Constraints</td>
<td>Computation Time (Sec.)</td>
<td>No. of Variables</td>
</tr>
<tr>
<td>1</td>
<td>3 x 3</td>
<td>33</td>
<td>9</td>
<td>20.73</td>
<td>33</td>
</tr>
<tr>
<td>2</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>35.26</td>
<td>&quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>28.83</td>
<td>&quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>26.63</td>
<td>&quot;</td>
</tr>
<tr>
<td>5</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>32.60</td>
<td>&quot;</td>
</tr>
<tr>
<td>6</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>44.03</td>
<td>&quot;</td>
</tr>
<tr>
<td>7</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>62.57</td>
<td>&quot;</td>
</tr>
<tr>
<td>8</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>100.13</td>
<td>&quot;</td>
</tr>
<tr>
<td>9</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>11.88</td>
<td>&quot;</td>
</tr>
<tr>
<td>10</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>68.55</td>
<td>&quot;</td>
</tr>
<tr>
<td>11</td>
<td>4 x 3</td>
<td>52</td>
<td>13</td>
<td>2098.67</td>
<td>52</td>
</tr>
<tr>
<td>12</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>257.66</td>
<td>&quot;</td>
</tr>
<tr>
<td>13</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>-</td>
<td>&quot;</td>
</tr>
<tr>
<td>14</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>56.16</td>
<td>&quot;</td>
</tr>
<tr>
<td>15</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>-</td>
<td>&quot;</td>
</tr>
</tbody>
</table>
Table 4.2  Computational Results for Line Balancing Problems

<table>
<thead>
<tr>
<th>Problem No.</th>
<th>Problem Size (Tasks)</th>
<th>pseudo-Boolean Program</th>
<th>DZLP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No.</td>
<td>No. of Variables</td>
<td>No. of Constraints</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>50</td>
<td>19</td>
</tr>
<tr>
<td>2</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>5</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>6</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>7</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>8</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>9</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>10</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
</tbody>
</table>
Table 4.3 Computational Results for Delivery Problems

<table>
<thead>
<tr>
<th>Problem No.</th>
<th>Problem Size ( n \times m )</th>
<th>pseudo-Boolean Program</th>
<th>DZLP</th>
<th>Computation Time (Sec.)</th>
<th>No. of Variables</th>
<th>No. of Constraints</th>
<th>Computation Time (Sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IBM 360/50</td>
<td></td>
<td></td>
<td>IBM 360/50</td>
</tr>
<tr>
<td>1</td>
<td>( 7 \times 3 )</td>
<td>7</td>
<td>3</td>
<td>0.70</td>
<td>7</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>1.30</td>
<td>&quot;</td>
<td>&quot;</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>0.68</td>
<td>&quot;</td>
<td>&quot;</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>1.35</td>
<td>&quot;</td>
<td>&quot;</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>0.71</td>
<td>&quot;</td>
<td>&quot;</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>( 14 \times 5 )</td>
<td>14</td>
<td>5</td>
<td>4.53</td>
<td>14</td>
<td>10</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>4.54</td>
<td>&quot;</td>
<td>&quot;</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>2.01</td>
<td>&quot;</td>
<td>&quot;</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>2.37</td>
<td>&quot;</td>
<td>&quot;</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>2.77</td>
<td>&quot;</td>
<td>&quot;</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>( 32 \times 6 )</td>
<td>32</td>
<td>6</td>
<td>29.43</td>
<td>32</td>
<td>12</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>53.60</td>
<td>&quot;</td>
<td>&quot;</td>
<td>-</td>
</tr>
</tbody>
</table>
### Table 4.4 Computational Results for Traveling Salesman Problems

<table>
<thead>
<tr>
<th>Problem No.</th>
<th>Problem Size (Cities)</th>
<th>No. of Variables</th>
<th>No. of Constraints</th>
<th>Computation Time (Sec.) IBM 360/50</th>
<th>No. of Variables</th>
<th>No. of Constraints</th>
<th>Computation Time (Sec.) IBM 360/50</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>24</td>
<td>12</td>
<td>4.00</td>
<td>24</td>
<td>18</td>
<td>63.30</td>
</tr>
<tr>
<td>2</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>5.16</td>
<td>&quot;</td>
<td>&quot;</td>
<td>64.46</td>
</tr>
<tr>
<td>3</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>4.17</td>
<td>&quot;</td>
<td>&quot;</td>
<td>67.99</td>
</tr>
<tr>
<td>4</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>3.89</td>
<td>&quot;</td>
<td>&quot;</td>
<td>106.54</td>
</tr>
<tr>
<td>5</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>8.75</td>
<td>&quot;</td>
<td>&quot;</td>
<td>136.28</td>
</tr>
<tr>
<td>6</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>5.40</td>
<td>&quot;</td>
<td>&quot;</td>
<td>148.67</td>
</tr>
<tr>
<td>7</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>3.91</td>
<td>&quot;</td>
<td>&quot;</td>
<td>235.15</td>
</tr>
<tr>
<td>8</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>3.93</td>
<td>&quot;</td>
<td>&quot;</td>
<td>72.47</td>
</tr>
<tr>
<td>9</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>3.84</td>
<td>&quot;</td>
<td>&quot;</td>
<td>165.77</td>
</tr>
<tr>
<td>10</td>
<td>&quot;</td>
<td>&quot;</td>
<td>&quot;</td>
<td>4.02</td>
<td>&quot;</td>
<td>&quot;</td>
<td>197.82</td>
</tr>
</tbody>
</table>
Table 4.5 Computational Results for Capital Allocation Problems

<table>
<thead>
<tr>
<th>Problem No.</th>
<th>pseudo-Boolean Program</th>
<th>DZLP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No. of Variables</td>
<td>No. of Constraints</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>5</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>6</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>7</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>8</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>9</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
</tbody>
</table>
Table 4.6 Computational Results for Fixed-Charge Problems

<table>
<thead>
<tr>
<th>Problem No.</th>
<th>Pseudo-Boolean Program</th>
<th>DZLP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No. of Variables</td>
<td>No. of Constraints</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>5</td>
<td>17</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>7</td>
<td>17</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>&quot;</td>
<td>&quot;</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>6</td>
</tr>
</tbody>
</table>
CHAPTER V

SUMMARY AND CONCLUSIONS

The combinatorial problem deals with the study of the arrangement of elements into sets. Whenever it is necessary to choose the best combination out of all possible arrangements, the problems are known as extremization problems. Various combinatorial problems such as shop scheduling, assembly-line balancing, delivery, traveling salesman, capital allocation and fixed-charge problem come under the category of extremization problems. In real situations, all the elements are integers and therefore these problems can be formulated as integer programming problems. By the proper utilization of zero-one variables, these problems can be converted into zero-one programming problems.

An algorithm proposed by Hammer and Rudeanu [35] is used to solve the zero-one programming problems. The algorithm makes use of the properties of pseudo-Boolean functions. A pseudo-Boolean function may be defined as a real-valued function $f(x_1, x_2, \ldots, x_n)$ with zero-one variables. A pseudo-Boolean program is a procedure to optimize a pseudo-Boolean function. The program uses a set of rules dependent on the properties of pseudo-Boolean functions. Using a branching and bounding procedure the search of all the branches is avoided. Improved results at each successive trial are utilized to improve the convergence to the optimum value.

Various combinatorial problems such as shop scheduling, assembly-line balancing, delivery, traveling salesman, capital allocation and
fixed-charge problems were formulated as zero-one programming problems.
The shop scheduling problem consists of \( J \) jobs to be performed on \( M \) machines in a prespecified machine ordering. The objective is to minimize the total completion time. The assembly-line balancing problem consists of minimizing the number of work stations for a constant cycle time. The delivery problem is concerned with the minimization of total shipping cost in fulfilling customer orders. The traveling salesman problem finds the route a traveling salesman should follow in visiting \( n \) cities so as to minimize the total distance traveled. In the capital allocation problem, a given amount of available investment should be so allocated to different projects so as to maximize the profit. In the fixed-charge problem, it is necessary to reduce the total cost involved while meeting the necessary requirements. All the above problems are similar in nature having linear objective functions, linear constraints and integer-valued variables. Hence all these problems can be formulated as zero-one programming problems.

The various combinatorial problems mentioned above were formulated as zero-one programming problems and were solved using the pseudo-Boolean programming. The same problems were solved using DZLP and the computational results were compared. In general the convergence of pseudo-Boolean program was better than DZLP for smaller and medium problems. The results of (3x3) flow shop problems show a marked reduction in computation time by using pseudo-Boolean program. Two (4x3) flow shop problems and all ten line balancing problems did not converge while using the pseudo-Boolean program. This was due to the poor fixation of values to the variables in various branches.
Due to the zero-one coefficients and unit right hand side in the constraint matrix, delivery problems converge to the optimal value rapidly while using the pseudo-Boolean program. The failure of DZLP in obtaining the solution of simple delivery problems came as a surprise. The reason for this failure could not be found out. Because of the assignment matrix constraints pseudo-Boolean program converges better than DZLP. Test problems in capital allocation and fixed-charge indicates the superiority of pseudo-Boolean program over DZLP in solving those problems.

The main draw back of the pseudo-Boolean program is the large amount of core locations it requires to store the node values of the branching tree. Hence pseudo-Boolean programming is a very efficient technique in solving small and medium sized problems.
REFERENCES


APPENDIX A

CONVERSION OF INTEGER PROGRAMMING PROBLEM TO
A ZERO-ONE FORM

The conversion of the integer linear programming to a zero-one form is discussed in this appendix. The conversion can be done either by the simple expansion technique [17] or by the Balas binary device [51]. For the conversion, it is necessary to know the upper bound on the value of each variable. In practical problems usually this upper bound is available.

A.1 The Simple Expansion Technique

For each integer valued variable \( x_j \) substitute zero-one variables \( x_{j1} \) such that

\[
x_j = x_{j1} + x_{j2} + \ldots + x_{juj}
\]

where \( U_j \) is the upper bound on the value of \( x_j \).

Consider an example in which it is required to minimize

\[
z = 2x_1 - 3x_2
\]

subject to

\[
X_1 - X_2 \geq 1
\]
\[
- x_1 \geq - 3
\]
\[
- x_2 \geq - 2
\]

and

\( X_1, X_2 \) non-negative integers.
The upper bound for $X_1$ is 3 from the second constraint and the upper bound for $X_2$ is 2 from the third constraints.

$$X_1 \leq U_1 = 3$$
$$X_2 \leq U_2 = 2$$

The following substitution is therefore made for the conversion of integer programming problem to zero-one form.

$$X_1 = x_{11} + x_{12} + x_{13}$$
$$X_2 = x_{21} + x_{22}$$

The problem now reduces to the following minimize

$$z = 2x_{11} + 2x_{12} + 2x_{13} - 3x_{21} - 3x_{22}$$

subject to

$$x_{11} + x_{12} + x_{13} - x_{21} - x_{22} \geq 1$$
$$- x_{11} - x_{12} - x_{13} \geq - 3$$
$$- x_{21} - x_{22} \geq - 2$$

and

all $x_{ij} = 0 \text{ or } 1$.

A.2 The Balas Binary Device.

Determine for each $X_j$ a value $L_j$ such that

$$L_j = \lfloor \log_2 U_j \rfloor + 1$$

where $U_j$ represents the upper bound on the value of the integer variable, $X_j$ and the bracket indicates the integer part of the quantity within the brackets. Then, for each $X_j$ substitute $L_j$ zero-one variables such that
\[ x_j = \sum_{k=1}^{L_j-k} 2^{x_{jk}}. \]

Thus considering the same general integer programming problem as before, we get

\[ L_1 = \lfloor \log_2 3 \rfloor + 1 \]
\[ = 1 + 1 \]
\[ = 2, \]

and

\[ L_2 = \lfloor \log_2 2 \rfloor + 1 \]
\[ = 1 + 1 \]
\[ = 2. \]

The following substitutions are therefore made for the conversion of integer programming problem to zero-one form.

\[ x_1 = 2x_{11} + x_{12} \]

\[ x_2 = 2x_{21} + x_{22} \]

The problem now reduces to the following

minimize

\[ z = 4x_{11} + 2x_{12} - 6x_{21} - 3x_{22} \]

subject to

\[ 2x_{11} + x_{12} - 2x_{21} - x_{22} \geq 1 \]

\[ - 2x_{11} - x_{12} \leq -3 \]

\[ - 2x_{21} - x_{22} \leq -2 \]

and

all \( x_{ij} = 0 \) or 1.

The conversion by the Balas binary device always results in less than or atmost equals to the number of zero-one variables than does
conversion by the simple expansion technique. Computationally, the Balas binary device expansion helps in the attainment of optimal solution in a shorter time than that of simple expansion technique.
APPENDIX B

COMPUTER PROGRAM LISTING

This appendix includes the computer program listing. The program is written in Fortran IV for IBM 360/50 computer. Data set 1 is used for input and logical unit 3 is used for printed output. The maximum number of variables and constraints are 60 and 25 respectively while using G Level Fortran. Program capacity can be changed by making changes in the DIMENSION statements. The computer program listing is shown on the following pages.
ILLEGIBLE

THE FOLLOWING DOCUMENT (S) IS ILLEGIBLE DUE TO THE PRINTING ON THE ORIGINAL BEING CUT OFF

ILLEGIBLE
LINEAR PSEUDO-BOOLEAN PROGRAMMING

PROGRAMMED BY
N. S. ANANTHA RANGA CHAR
DEPARTMENT OF INDUSTRIAL ENGINEERING
KANSAS STATE UNIVERSITY

BASED ON THE ALGORITHM PROPOSED BY
PETER L. HAMMER & SERGIU RUIDANU

********************************************************************************************

VARIABLES EXPLANATION:
________________________________________________________________________________________

******* INPUT VARIABLES *******

NPROB  NUMBER OF PROBLEMS
N      NUMBER OF VARIABLES
M      NUMBER OF CONSTRAINTS (INCLUDING OBJ. FN.)
NPRINT EQUALS 0 IF NODE VALUES ARE NOT TO BE PRINTED
       EQUALS 1 IF NODE VALUES ARE TO BE PRINTED
ISTART EQUALS 2 FOR SCHEDULING, LINE BALANCING
         AND FIXED CHARGE PROBLEMS
         EQUALS 1 FOR TRAVELING SALESMAN, DELIVERY
         AND CAPITAL ALLOCATION PROBLEMS
NPOINT EQUALS 0 FOR OBTAINING MULTIPLE OPTIMUM
         POINTS
         EQUALS 1 FOR OBTAINING SINGLE OPTIMUM
         POINT
ND(I)   RIGHT HAND SIDE OF THE CONSTRAINTS WITH
         ND(1) AS UPPER BOUND ON THE OBJ. FN.
NC(I,J) COEFFICIENT MATRIX (INCLUDING OBJ. FN.)
NTYPE(I) EQUALS 1 IF CONSTRAINT IS OF TYPE (=EQ)
         EQUALS 2 IF CONSTRAINT IS OF TYPE (=GE)
         EQUALS 3 IF CONSTRAINT IS OF TYPE (=LE)
NTYPE(J) EQUALS 2 FOR MAXIMIZATION PROBLEMS
         EQUALS 3 FOR MINIMIZATION PROBLEMS

******* PROGRAM VARIABLES *******

NZCOL(J) ZERO-ONE VARIABLE X(J) WHOSE VALUE IS
         TO BE DETERMINED
N7(I,J) INDEX USED TO KEEP TRACK WHETHER THE
         VARIABLE X(J) IS POSITIVE OR NEGATIVE
         EQUALS 3 INDICATE POSITIVE X(J)
         EQUALS 4 INDICATE NEGATIVE X(J)
KPOINT LEVEL INDICATOR
KP(J),NP(J) INDICATORS TO CHECK WHETHER A BRANCH IS
            EXPLORED OR NOT
NOBJ(J) STORED VALUE OF THE VARIABLE X(J)
        WITH X(J)=2 PERTAINING TO FREE VARIABLES
IMAX(KPOINT) CONSTRAINT USED FOR DETERMINING
             BRANCHING AT LEVEL KPOINT
JMAX(KPOINT) VARIABLE X(J) USED FOR DETERMINING
             BRANCHING AT LEVEL KPOINT
C  K  INDICATOR TO TO NOTE WHETHER ANY VARIABLE
C  HAS BEEN DETERMINED IN A SEARCH OR NOT
C
C******************************************************************************
C  INPUT INSTRUCTIONS :
C******************************************************************************
C  CARD 1  NPROB,NPRINT, ISTART,NPOINT  FORMAT(16I5)
C  CARD 2  N,M  FORMAT(16I5)
C  CARD 3  (NTYPE(I),I#1,M)  FORMAT(16I5)
C  CARD 4  (IND(I),I=1,M)  FORMAT(16I5)
C  CARD 5  CONSTRAINT MATRIX STARTING FROM OBJECTIVE
C           FUNCTION. START EACH CONSTRAINT IN NEW
C           ROW  FORMAT(16I5)
C  REPEAT FROM CARD 2 FOR EACH PROBLEM
C  PROGRAM HALTS AFTER EXECUTING NPROB NUMBER OF PROBLEMS
C******************************************************************************
C  MAIN PROGRAM
C******************************************************************************

OC01  IMPLICIT INTEGER*2(I-N)
OC02  INTEGER*4 A1, A2
OC03  COMMON M,N,I,K,KPOINT,NPRINT
OC04  COMMON NZ(25,60),NC(25,60),NO(25),NTYPE(25),
      1NZSTR(60,60),NCSTR(60,25,60),NZCOL(60),IMAX(25),
      2JMAX(25),NOBJ(60),NDSTR(25,60),NP(60),KP(60),NSOLN(60)

OC05  FORMAT(1H1,25I5)
OC06  1 FORMAT(1HO,' THE MINIMIZING POINTS ARE GIVEN BY')
OC07  2 FORMAT(1HO,' THE NEW BOUND ON THE OBJECTIVE FUNCTION='15)
OC08  3 FORMAT(1HO,' THE NEW VALUE OF THE OBJECTIVE FUNCTION='15)
OC09  4 FORMAT(1HO,' SEARCH IS OVER',//)
OC10  5 FORMAT(1HO,'PROBLEM NUMBER =',I3,40(1H*))
OC11  6 FORMAT(1HO,'MINIMUM VALUE OF THE OBJECTIVE FUNCTION',
      1I10)
OC12  7 FORMAT(1H1)
OC13  8 FORMAT(1HO,'DATA INPUT TO THE PROBLEM')
OC14  9 FORMAT(1HO,'RIGHT HAND SIDE')
OC15 10 FORMAT(1HO,'THE COEFFICIENT MATRIX')
OC16 11 FORMAT(1HO,'UPDATED BOUND AT LEVEL = '15,15X,15)
OC17 12 FORMAT(1HO,'MODIFIED OBJECTIVE FUNCTION')
OC18 13 FORMAT(1HO,'ACCELERATING TEST INDICATES TERMINATION IN'
      1' THIS BRANCH X('12,')')
OC19 14 FORMAT(1HO,'TIME TAKEN FOR COMputation =',F7.2,
      1' SECONDS')
OC20 15 FORMAT(1HO,'BRANCHING POINT GOING ABOVE UPPER LIMIT'
      110X'CHECK FOR ERROR')
OC21 16 FORMAT(1HO,'END OF DATA WHILE READING N AND M IN PROBL'
      1'EM NO.'13)
OC22 17 FORMAT(1HO,'ERROR ENCOUNTERED WHILE READING N AND M IN'
      1'PROBLEM NO.'13)
OC23 18 FORMAT(1HO,'END OF DATA WHILE READING NTYPE IN PROBLEM'
      1'NO.'13)
OC24 19 FORMAT(1HO,'ERROR ENCOUNTERED WHILE READING NTYPE IN '
1 'PROBLEM NO., 13')
0025 26 FORMAT(1HO,'END OF DATA WHILE READING RHS IN PROBLEM '
0026 1 'NO., 13')
0027 27 FORMAT(1HO,'ERROR ENCOUNTERED WHILE READING RHS IN PRO'
0028 1 'BLEM NO., 13')
0029 28 FORMAT(1HO,'END OF DATA WHILE READING COEFFICIENTS IN'
0030 1 'EQUATION ',I3,'OF PROBLEM',I3)
0031 29 FORMAT(1HO,'ERROR ENCOUNTERED WHILE READING COEFFICE'
0032 1 'NTS IN EQUATION ',I3,'OF PROBLEM',I3)
0033 30 FORMAT(1HO,120(1H*))
0034 31 FORMAT(1HO,'LINEAR PSEUDO-BOOLEAN PROGRAMMING'//1IX
0035 1 'PROGRAMED BY')
0036 32 FORMAT(1HO,10X,'N. S. ANANTHA RANGA CHAR.')
0037 33 FORMAT(1HO,10X,'DEPT. OF INDUSTRIAL ENGINEERING')
0038 34 FORMAT(1HO,10X,'KANSAS STATE UNIVERSITY')
0039 35 FORMAT(1HO,20X'BASED ON THE ALGORITHM PROPOSED BY '
0040 1 'PETER L. HAMMER & SERGIU RUDEANU')
0041 41 FORMAT(16I5)
0042 C
0043 C  READ THE DATA CARDS
0044 C  READ(1,41) NPROB,NPRINT,ISTART,NPOINT
0045 NPRB=1
0046 45 WRITE(3,6) NPRB
0047 CALL TIME(AT1)
0048  READ(1,41,END=700,ERR=705) N,M
0049 READ(1,41,END=710,ERR=715)(NTYPE(I),I=1,M)
0050 DO 50 I=1,M
0051 50 READ(1,41,END=730,ERR=735)(NC(I,J),J=1,N)
0052 C  PRINT OUT THE DATA
0053 C  WRITE(3,9)
0054 WRITE(3,10)
0055 WRITE(3,11)
0056 DO 52 I=1,M
0057 WRITE(3,8)
0058 52 WRITE(3,1)(NC(I,J),J=1,N)
0059 C  IF PROBLEM IS MAXIMIZATION CHANGE TO MINIMIZATION
0060 IF(NTYPE(I),NE.2) GO TO 53
0061 DO 49 J=1,N
0062 49 NC(I,J)=NC(I,J)
0063 ND(1)=-ND(1)
0064 NTYPE(1)=3
0065 C  IF INEQUALITY IS OF TYPE (LE.) MAKE IT OF TYPE (GE.)
C
53 DO 54 J=1,N
54 NOBJ(J)=NC(1,J)
55 DO 58 I=1,M
57 IF(NTYPE(I).NE.3) GO TO 58
58 J=1,N
59 NC(I,J)=-NC(I,J)
60 NTYPE(I)=2
61 CONTINUE
C
C INITIALISE THE VALUES AND STORE THE OBJECTIVE FUNCTION
C
73 NOLD=10000
74 NN=N+1
75 NRHS=0
76 DO 59 J=1,N
77 IF(NC(1,J).LT.0) NRHS=NRHS-NC(1,J)
78 NP(J)=2
79 KP(J)=2
80 59 NZCOL(J)=2
C
C ELIMINATE THE NEGATIVE SIGNS
C
81 DO 100 I=1,M
82 DO 90 J=1,N
83 NZ(I,J)=3
84 IF(NC(I,J).GE.0) GO TO 90
85 NC(I,J)=-NC(I,J)
86 NZ(I,J)=4
87 ND(I)=ND(I)+NC(I,J)
88 90 CONTINUE
89 100 CONTINUE
C
C PRINT MODIFIED EQUATIONS IF NPRINT.EQ.1
C
90 IF(NPRINT.EQ.0) GO TO 214
91 WRITE(3,13)
92 WRITE(3,10)
93 WRITE(3,1)(ND(I),I=1,M)
94 WRITE(3,11)
95 DO 212 I=1,M
96 WRITE(3,1)(NZ(I,J),J=1,N)
97 212 WRITE(3,1)(NC(I,J),J=1,N)
C
C STORE THE STARTING VALUES
C
98 214 KPOINT=0
99 CALL RECORD(81000)
100 IF(KPOINT.GT.NN) GO TO 900
C
C SELECT THE BRANCH POINT
C
101 MAX=0
102 IF(ISTART.EQ.0) ISTART=1
103 250 DO 270 I=ISTART,M
104 255 DO 260 J=1,N
105 IF(NC(I,J).LE.MAX) GO TO 260
MAX=NC(I,J)
IMAX(KPOINT)=I
JMAX(KPOINT)=J
260 CONTINUE
IF(MAX.GT.0) GO TO 272
270 CONTINUE
IMAX1=IMAX(KPOINT)
IMAX2=JMAX(KPOINT)
275 IF(NZ(1,IMAX2).EQ.3) GO TO 350
C IF NZ=ZBAR SUBSTITUTE Z=0
276 NZCOL(IMAX2)=0
NP(KPOINT)=0
DO 280 INDEX=1,M
IF(NZ(INDEX,IMAX2).EQ.3) GO TO 280
ND(INDEX)=ND(INDEX)-NC(INDEX,IMAX2)
280 NC(INDEX,IMAX2)=0
C SUBSTITUTE THE BRANCH VALUES IN ALL CONSTRAINTS
K=0
DO 300 I=1,M
IF(NTYPE(I).NE.1) GO TO 290
285 CALL EQUAL(&310)
281 IF(K.EQ.1) GO TO 281
300 CONTINUE
IF(K.EQ.1) GO TO 281
GO TO 245
C IF Z=0 FAILS TRY Z=1
310 CALL UPDATE(&1000)
311 NZCOL(IMAX2)=1
KP(KPOINT)=1
315 DO 315 INDEX=1,M
310 IF(NZ(INDEX,IMAX2).EQ.4) GO TO 315
317 ND(INDEX)=ND(INDEX)-NC(INDEX,IMAX2)
315 NC(INDEX,IMAX2)=0
C SUBSTITUTE THE BRANCH VALUES IN ALL CONSTRAINTS
K=0
DO 340 I=1,M
IF(NTYPE(I).NE.1) GO TO 330
320 CALL EQUAL(&345)
320 IF(K.EQ.1) GO TO 316
340 CONTINUE
330 CALL INEQOL(&345)
340 CONTINUE
IF(K.EQ.1) GO TO 316
GO TO 245
C IF Z=0 & Z=1 FAILS GO ONE LEVEL DOWN AND
C CHANGE THE BRANCH
C
0149     345 KPOINT=KPOINT-1
0150     IF(KPOINT.LT.1) GO TO 1000
0151     IF((NP(KPOINT).EQ.0).AND.(KP(KPOINT).EQ.1)) GO TO 345
0152     IMAX1=IMAX(KPOINT)
0153     IMAX2=JMAX(KPOINT)
0154     NDUMMY=NCOL(IMAX2)
0155     CALL UPDATE(&1000)                      
0156     IF(NDUMMY.EQ.0) GO TO 311               
0157     GO TO 376
C
C     IF NZ=Z SUBSTITUTE Z=1
C
0158     350 NZCOL(IMAX2)=1                       
0159     KP(KPOINT)=1                            
0160     DO 355 INDEX=1,M                        
0161     IF(NZ(INDEX,IMAX2).EQ.4) GO TO 355      
0162     ND(INDEX)=ND(INDEX)-NC(INDEX,IMAX2)     
0163     355 NC(INDEX,IMAX2)=0                    
C
C     SUBSTITUTE THE BRANCH VALUES IN ALL CONSTRAINTS
C
0164     356 K=0                                  
0165     DO 370 I=1,M                            
0166     IF(NTYPE(I).NE.1) GO TO 365              
0167     360 CALL EQUAL(&375)                    
0168     IF(K.EQ.1) GO TO 356                    
0169     GO TO 370                               
0170     365 CALL INEQL(&375)                    
0171     370 CONTINUE                            
0172     IF(K.EQ.1) GO TO 356                    
0173     GO TO 245                               
C
C     IF Z=1 FAILS TRY Z=0
C
0174     375 CALL UPDATE(&1000)                  
0175     376 NZCOL(IMAX2)=0                      
0176     KP(KPOINT)=0                           
0177     DO 380 INDEX=1,M                       
0178     IF(NZ(INDEX,IMAX2).EQ.3) GO TO 380      
0179     ND(INDEX)=ND(INDEX)-NC(INDEX,IMAX2)    
0180     380 NC(INDEX,IMAX2)=0                   
C
C     SUBSTITUTE THE BRANCH VALUES IN ALL CONSTRAINTS
C
0181     381 K=0                                 
0182     DO 400 I=1,M                           
0183     IF(NTYPE(I).NE.1) GO TO 390             
0184     385 CALL EQUAL(&410)                   
0185     IF(K.EQ.1) GO TO 381                   
0186     GO TO 400                              
0187     390 CALL INEQL(&410)                   
0188     400 CONTINUE                           
0189     IF(K.EQ.1) GO TO 381                   
0190     GO TO 245                             
C
C     IF Z=1 & Z=0 FAILS GO ONE LEVEL DOWN AND
C     CHANGE THE BRANCH
C
0191 410 KPOINT=KPOINT-1
0192 IF(KPOINT.LT.1) GO TO 1000
0193 IF((NP(KPOINT),EQ.0).AND.(KP(KPOINT),EQ.1)) GO TO 410
0194 IMAX1=IMAX(KPOINT)
0195 IMAX2=JMAX(KPOINT)
0196 NDUMMY=NZCOL(IMAX2)
0197 CALL UPDATE(@1000)
0198 IF(NDUMMY.EQ.0) GO TO 311
0199 GO TO 376
C
C ESTABLISH NEW BOUND ON THE OBJECTIVE FUNCTION
C
0200 450 DO 460 J=1,N
0201 460 NSOLN(J)=NZCOL(J)
0202 NEW=0
0203 DO 500 J=1,N
0204 500 NEW=NEW+NOBJ(J)*NZCOL(J)
0205 OLD=NEW
0206 NADD=-NEW+NRHS-NDSTR(1,1)+NPOINT
C
C PRINT THE FEASIBLE SOLUTION
C
0207 WRITE(3,2)
0208 WRITE(3,1)(NSOLN(J),J=1,N)
0209 WRITE(3,4)NEW
0210 WRITE(3,8)
C
C CONTINUE THE SEARCH
C
0211 KPOINT=KPOINT-1
0212 IF(KPOINT.LT.1) GO TO 1000
0213 DO 509 K=1,KPOINT
0214 509 NDSTR(1,K)=NDSTR(1,K)+NADD
0215 IF(NPRINT.NE.0) WRITE(3,12) ((K,NDSTR(1,K)),K=1,KPOINT)
0216 GO TO 515
0217 510 KPOINT=KPOINT-1
0218 IF(KPOINT.LT.1) GO TO 1000
0219 IMAX1=IMAX(KPOINT)
0220 IMAX2=JMAX(KPOINT)
0221 NDUMMY=NZCOL(IMAX2)
0222 CALL UPDATE(@1000)
0223 IF((NP(KPOINT),EQ.0).AND.(KP(KPOINT),EQ.1)) GO TO 510
C
C ACCELERATION TEST
C
0224 530 NSUM=0
0225 DO 540 J=1,N
0226 540 IF(NZCOL(J).NE.2) GO TO 540
0227 IF(J.EQ.IMAX2) GO TO 540
0228 IF((NZ(1,J),EQ.3).AND.(NSOLN(J),EQ.0))
0229 INSUM=NSUM+NCSTR(1,1,J)
0230 540 CONTINUE
0231 IF(NC(1,IMAX2).LE.NSUM) GO TO 545
0232 IF(NPRINT.NE.0) WRITE(3,14) IMAX2
0233 GO TO 510
545 IF(NDUMMY.EQ.0) GO TO 311
GO TO 376

C PRINT ERROR MESSAGES IN CASE OF ERROR IN DATA

700 WRITE(3,22) NPRB
GO TO 1001
705 WRITE(3,23) NPRB
GO TO 1001
710 WRITE(3,24) NPRB
GO TO 1001
715 WRITE(3,24) NPRB
GO TO 1001
720 WRITE(3,26) NPRB
GO TO 1001
725 WRITE(3,27) NPRB
GO TO 1001
730 WRITE(3,28) I,NPRB
GO TO 1001
735 WRITE(3,29) I,NPRB
GO TO 1001

C SEARCH IS OVER. PRINT THE RESULT AND TIME TAKEN.

900 WRITE(3,21)
1000 WRITE(3,5)
WRITE(3,30)
WRITE(3,2)
WRITE(3,1)(NSOLN(J),J=1,N)
WRITE(3,7) NEW
CALL TIME(AT2)
TTIME=(AT2-AT1)/100.
WRITE(3,20) TTIME
WRITE(3,30)
NPRB=NPRB+1
IF(NPRB.LE.NPROB) GO TO 45
STOP
END
SUBROUTINE EQUAL(*)

C******************************************************************************
C THIS SUBROUTINE COMPUTES THE VALUE OF THE VARIABLE
C APPEARING IN EQUALITY CONSTRAINTS.
C THE ROUTINE TESTS WHETHER ANY EQUATION SATISFIES
C DETERMINE CASES.
C IF ANY DETERMINE CASE IS SATISFIED THE VALUE IS
C FIXED ACCORDING TO THE PARTICULAR CASE.
C******************************************************************************

IMPLICIT INTEGER*2(I-N)
COMMON M,N,I,K,KPOINT,NPRINT
COMMON NZ(25,60),NC(25,60),ND(25),NTYPE(25),
INZ(60,25,60),NCG(60,25,60),NZCOL(60),IMAX(25),
2JMAX(25),NOBJ(60),NDSTR(25,60),NP(60),KP(60),NSOLN(60)

2 FORMAT(' NO SOLUTION IN THIS BRANCH, CHANGE THE BRANCH')

IF(ND(I).*GE.*0) GO TO 5

IF(PROGRAM**ENTERS**THIS**POINT**IT**IS**CASE**1

IF(NPRINT.*NE.*0) WRITE(3,2)
RETURN1

5 NSUM=0
CD 6 J=1,N
6 NSUM=NSUM+NC(I,J)
10 IF((NSUM.*EQ.*0).AND.(ND(I).*EQ.*0)) RETURN

C THIS IS CASE 2

CALL ENTRY1
DO 20 INDEX=1,M
20 IF((NTYPE(INDEX).*EQ.*1).AND.(ND(INDEX).*LT.*0)) RETURN1
K=1
GO TO 100
25 NSUM=0
CD 30 J=1,N
30 NSUM=NSUM+NC(I,J)
32 IF((NSUM.*EQ.*0)) RETURN1
31 IF((NSUM.*GE.*ND(I))) GO TO 40

C THIS IS CASE 5

IF(NPRINT.*NE.*0) WRITE(3,2)
RETURN1

40 IF((NSUM.*GT.*ND(I))) GO TO 50

C THIS IS CASE 6

CALL ENTRY2
DO 45 INDEX=1,M
45 IF((NTYPE(INDEX).*EQ.*1).AND.(ND(INDEX).*LT.*0)) RETURN1
K=1
GO TO 100
USE CASE 3 FOR ANY VARIABLE IF IT APPLIES

50 DO 80 J=1,N
   IF(NZ(I,J).LE.ND(I)) GO TO 80
0034   IF(NZ(I,J).EQ.3)GO TO 65
0035   55 NZCOL(J)=1
0036   DO 60 INDEX=1,M
0037      IF(NZ(INDEX,J).EQ.4) GO TO 60
0038      ND(INDEX)=ND(INDEX)-NC(INDEX,J)
0039   60 NC(INDEX,J)=0
0040   K=1
0041   GO TO 80
0042   65 NZCOL(J)=0
0043   DO 70 INDEX=1,M
0044      IF(NZ(INDEX,J).EQ.3)GO TO 70
0045      ND(INDEX)=ND(INDEX)-NC(INDEX,J)
0046   70 NC(INDEX,J)=0
0047   K=1
0048   80 CONTINUE
0049   NSUM=0
0050   DO 85 J=1,N
0051   85 NSUM=NSUM+NC(I,J)
0052   IF((NSUM.EQ.0).AND.(ND(I).NE.0))RETURN1
0053   IF(NSUM.EQ.0) GO TO 100
0054   IF(NSUM.LE.ND(I)) GO TO 31
0055   NDUMMY=NC(I,1)
0056   IND=1
0057   DO 90 J=2,N
0058   IF(NDUMMY.GE.NC(I,J)) GO TO 90

TRY CASE 7

NDUMMY=NC(I,J)
0060   IND=J
0061   90 CONTINUE
0062   NSUM1=NSUM-NDUMMY
0063   IF(NSUM1.GE.ND(I)) GO TO 100
0064   IF(NZ(I,IND).EQ.3)GO TO 95
0065   NZCOL(IND)=0
0066   DO 94 INDEX=1,M
0067      IF(NZ(INDEX,IND).EQ.3) GO TO 94
0068      ND(INDEX)=ND(INDEX)-NC(INDEX,IND)
0069   94 NC(INDEX,IND)=0
0070   K=1
0071   GO TO 5
0072   95 NZCOL(IND)=1
0073   DO 99 INDEX=1,M
0074      IF(NZ(INDEX,IND).EQ.4) GO TO 99
0075      ND(INDEX)=ND(INDEX)-NC(INDEX,IND)
0076   99 NC(INDEX,IND)=0
0077   K=1
0078   GO TO 5
0079   100 RETURN
0080   END
SUBROUTINE INEQL(*)
C******************************************************************************
C THIS SUBROUTINE COMPUTES THE VALUE OF THE VARIABLE
C APPEARING IN INEQUALITY CONSTRAINTS.
C THE ROUTINE TESTS WHETHER ANY INEQUALITY SATISFIES
C DETERMINE CASES.
C IF ANY DETERMINE CASE IS SATISFIED THE VALUE IS
C FIXED ACCORDING TO THE PARTICULAR CASE.
C******************************************************************************
C IMPLICIT INTEGER*2(I-N)
0003 COMMON M,N,I,K,KPOINT,NPRINT
0004 COMMON NZ(25,60),NC(25,60),ND(25),NTYPE(25),
1 INZSTR(60,60),NCSTR(60,25,60),NZCOL(60),IMAX(25),
2 JMAX(25),NOBJ(60),NDSTR(25,60),NP(60),KP(60),NSOLN(60)
C 2 FORMAT(' NO SOLUTION IN THIS BRANCH,CHANGE THE BRANCH')
0006 IF(ND(I),GT.0) GO TO 10
C THIS IS REDUNDANT INEQUALITY , CASE 1
0007 GO TO 90
C TEST FOR CASE 2
C
0008 10 DO 20 J=1,N
0009 IF(NC(I,J),GE,ND(I)) GO TO 90
0010 20 CONTINUE
0011 25 NSUM=0
0012 DO 30 J=1,N
0013 30 NSUM=NSUM+NC(I,J)
0014 IF(NSUM,EQ.0)RETURN
C TEST FOR CASE 3
C
0015 IF(NSUM,GE,ND(I)) GO TO 40
0016 IF(NPRINT,NE.0) WRITE(3,2)
0017 RETURN
C TEST FOR CASE 4
C
0018 40 IF(NSUM,GT,ND(I)) GO TO 50
0019 CALL ENTRY2
0020 DO 45 INDEX=1,M
0021 45 IF((NTYPE(INDEX),EQ.1).AND.(ND(INDEX),LT.0)) RETURN
0022 K=1
0023 GO TO 90
C TEST FOR CASE 5
C
0024 50 NDUMMY=NC(I,1)
0025 IND=1
0026 DO 60 J=2,N
0027 IF(NDUMMY,GE,NC(I,J)) GO TO 60
0028 NDUMMY=NC(I,J)
0029 IND=J
60 CONTINUE
NSUM1=NSUM-NDUMMY
IF(NSUM1.GE.ND(I)) GO TO 90
IF(NZ(I,IND).EQ.3) GO TO 75
65 NZCOL(IND)=0
DO 70 INDEX=1,M
IF(NZ(INDEX,IND).EQ.3) GO TO 70
ND(INDEX)=ND(INDEX)-NC(INDEX,IND)
70 NC(INDEX,IND)=0
K=1
GO TO 10
75 NZCOL(IND)=1
DO 80 INDEX=1,M
IF(NZ(INDEX,IND).EQ.4) GO TO 80
ND(INDEX)=ND(INDEX)-NC(INDEX,IND)
80 NC(INDEX,IND)=0
K=1
GO TO 10
90 RETURN
END
SUBROUTINE RECORD(*)

C*****************************************************************************
C THIS SUBROUTINE KEEPS TRACK OF THE VALUE OF
C THE VARIABLES APPEARING AT ALL BRANCH POINTS.
C THE LEVELS ARE INDICATED BY THE VARIABLE 'KPOINT'
C*****************************************************************************

IMPLICIT INTEGER*2(I-N)
COMMON M,N,I,K,KPOINT,NPRINT
COMMON NZ(25,60),NC(25,60),ND(25),NTYPE(25),
  1NZSTR(60,60),NCSTR(60,25,60),NZCOL(60),IMAX(25),
  2JMAX(25),NOBJ(60),NDSTR(25,60),NP(60),KP(60),NSOLN(60)

1 FORMAT(1HO'STORED VALUES')
2 FORMAT(1HO'LEVEL = 'I2)
3 FORMAT(1HO'VALUES OF NZ STORED')
4 FORMAT(1H ,25I5)

KPOINT=KPOINT+1
5 DO 15 J=1,N
  6 DO 10 I=1,M
  7 NDSTR(I,KPOINT)=ND(I)
  8 NCSTR(KPOINT,I,J)=NC(I,J)
  9 NZSTR(KPOINT,J)=NZCOL(J)
10 IF(NPRINT.EQ.0) GO TO 25
11 WRITE(3,1)
12 WRITE(3,2)KPOINT
13 WRITE(3,3)
14 WRITE(3,4)(NZSTR(KPOINT,J),J=1,N)
15 RETURN
16 ENTRY UPDATE(*)

C*****************************************************************************
C THIS ROUTINE SUPPLIES THE VALUE OF THE VARIABLES
C STORED AT DIFFERENT BRANCH POINTS .
C THE MAIN PROGRAM SUPPLIES THE LEVEL 'KPOINT'
C AT WHICH THE VALUES ARE REQUIRED
C*****************************************************************************

6 FORMAT(1HO'THE VALUES ARE UPDATED TO THE LEVEL = 'I5)
7 FORMAT(1HO'VALUES OF NZ')
8 FORMAT(1H ,25I5)

IF(KPOINT.LT.1) RETURN1
9 DO 40 J=1,N
10 DO 30 I=1,M
11 ND(I)=NDSTR(I,KPOINT)
12 NC(I,J)=NCSTR(KPOINT,I,J)
13 NZCOL(J)=NZSTR(KPOINT,J)
14 IF(NPRINT.EQ.0) GO TO 55
15 WRITE(3,6)KPOINT
16 WRITE(3,7)
17 WRITE(3,8)(NZCOL(J),J=1,N)
18 LP=KPOINT+1
19 DO 60 K=LP,N,1
20 NP(K)=2
21 KP(K)=2
22 RETURN1
RETURN
END
SUBROUTINE ENTRY1

C***********************************************************************
C THIS SUBROUTINE FIXES UP THE VALUES OF ALL VARIABLES
C IF THE EQUALITY CONSTRAINT SATISFIES CASE 2
C***********************************************************************

C IMPLICIT INTEGER*2(I-N)
COMMON M,N,I,K,KPOINT,NPRINT
COMMON NZ(25,60),NC(25,60),ND(25),NTYPE(25),
INZSTR(60,60),NCSTR(60,25,60),NZCOL(60),IMAX(25),
2JMAX(25),NOBJ(60),NDSTR(25,60),NP(60),KP(60),NSOLN(60)

10 DO 40 J=1,N
   IF(NC(J,J).EQ.0) GO TO 40
   IF(NZ(J,J).EQ.3) GO TO 25
   15 NZCOL(J)=1
   DO 20 INDEX=1,M
   20 NC(INDEX,J)=0
   GO TO 40
   DO 30 INDEX=1,M
   IF(NZ(INDEX,J).EQ.3) GO TO 30
   ND(INDEX)=ND(INDEX)-NC(INDEX,J)
   30 NC(INDEX,J)=0
   CONTINUE
   RETURN
END
SUBROUTINE ENTRY2
IMPLICIT INTEGER*2(I-N)

C
C******************************************************************************
C THIS SUBROUTINE FIXES UP THE VALUES OF ALL VARIABLES
C IF EQUALITY CONSTRAINT SATISFFIES CASE 6 OR
C THE INEQUALITY CONSTRAINT SATISFFIES CASE 4
C******************************************************************************
C
COMMON M,N,I,K,KPOINT,NPRINT
COMMON NZ(25,60),NC(25,60),ND(25),NTYPE(25),
INZSTR(60,60),NCSTR(60,25,60),NZCOL(60),IMAX(25),
2JMAX(25),NOBJ(60),NDSTR(25,60),NP(60),KP(60),NSOLN(60)

10 DO 40 J=1,N
  IF(NC(I,J)==0) GO TO 40
  IF(NZ(I,J)==4) GO TO 25
  NZCOL(J)=1
  DO 20 INDEX=1,M
    IF(NZ(INDEX,J)==4) GO TO 20
    ND(INDEX)=ND(INDEX)-NC(INDEX,J)
  20 NC(INDEX,J)=0
  GO TO 40

15 NZCOL(J)=0
  DO 30 INDEX=1,M
    IF(NZ(INDEX,J)==3) GO TO 30
    ND(INDEX)=ND(INDEX)-NC(INDEX,J)
  30 NC(INDEX,J)=0
  CONTINUE
R
RETURN
END
DATA INPUT TO THE PROBLEM

RIGHT HAND SIDE
\[ C = -18 \quad -15 \quad 0 \quad 0 \]

THE COEFFICIENT MATRIX
\[
\begin{bmatrix}
-4 & -2 & -1 & -4 & -2 & -1 & -4 & -2 & -1 & 0 & 0 \\
-4 & -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 \\
0 & 0 & 0 & -4 & -2 & -1 & 0 & 0 & 0 & 0 & 7
\end{bmatrix}
\]

THE MINIMIZING POINTS ARE GIVEN BY
\[ C = 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \]

THE NEW VALUE OF THE OBJECTIVE FUNCTION = -7

SEARCH IS OVER

THE MINIMIZING POINTS ARE GIVEN BY
\[ C = 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \]

MINIMUM VALUE OF THE OBJECTIVE FUNCTION = -7

TIME TAKEN FOR COMPUTATION = 8.55 SECONDS
APPLICATION OF LINEAR PSEUDO-BOOLEAN
PROGRAMMING TO COMBINATORIAL PROBLEMS

by

NADIPURAM SREERANGA CHAR ANANTHA RANGA CHAR

B.E. (Mech.), University of Mysore, India, 1965

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Industrial Engineering

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1970
The combinatorial problems deal with the study of the arrangement of elements into sets. Whenever it is necessary to choose the best combination out of all possible arrangements, the problems are known as extremization problems. Various combinatorial problems such as shop scheduling, assembly-line balancing, delivery, traveling salesman, capital allocation and fixed-charge problem come under the category of extremization problems. These problems are similar in nature, having linear objective functions, linear constraints and integer-valued variables. Therefore these problems can be formulated as integer programming problem. By the proper utilization of zero-one variables, these problems can be converted into zero-one programming problems.

The linear pseudo-Boolean algorithm proposed by Hammer and Rudeanu is used to solve the zero-one programming problems. The program uses a set of rules dependent on the properties of pseudo-Boolean functions. Using a branching and bounding procedure the search is restricted to a limited number of branches. Improved results at each trial are utilized successively to improve the convergence to optimum value.

The various combinatorial problems mentioned above were formulated as zero-one programming problems and were solved using the pseudo-Boolean programming. The same problems were solved using IBM program DZLP developed by Salskin and Spielburg. In general, the convergence of pseudo-Boolean program was better than that of DZLP for small and medium-sized problems. Two (4x3) flow-shop problems and the line balancing problems did not converge while using the pseudo-Boolean program. DZLP failed in obtaining the solution of simple delivery problems.
The main drawback of the pseudo-Boolean program is the large amount of core storage it requires for the node values of the branching tree. Hence pseudo-Boolean programming is a very efficient technique in solving small and medium-sized problems.