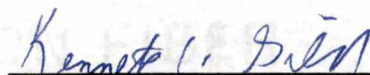


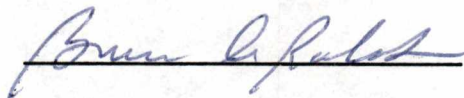
To the Graduate Council:

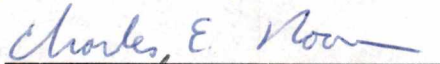
I am submitting herewith a dissertation written by Arup Kumar Mukherjee entitled "Active Resource Scheduling with Equivalent Tasks, Unary Demands and Hierarchical Structure Capacity Constraints." I have examined the final copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Management Science.

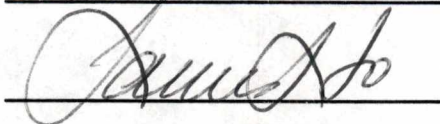


Kenneth C. Gilbert, Major Professor

We have read this dissertation
and recommend its acceptance:







Accepted for the Council:



Vice Provost
and Dean of the Graduate School

ACTIVE RESOURCE SCHEDULING WITH EQUIVALENT TASKS,
UNARY DEMANDS AND HIERARCHICAL STRUCTURE
CAPACITY CONSTRAINTS

A Dissertation
Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Arup Kumar Mukherjee

May, 1991

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and appreciation to Dr. Kenneth C. Gilbert, my dissertation adviser, for his expert guidance, encouragement, and patience in the completion of this dissertation. Very special thanks to Dr. Gilbert for teaching me the art of technical report writing. I acknowledge with gratitude the expert guidance of Dr. Charles E. Noon during different stages of this dissertation and throughout my graduate program. I would also like to thank sincerely each of the rest of my doctoral committee for his continued interest and help: Dr. James K. Ho and Dr. Bruce A. Ralston.

Very special thanks to Dr. Oscar F. Fowler for receiving his continuous encouragement throughout my graduate program. I acknowledge with pleasure the guidance and patience of Mr. Steve Martin during development of the decision support system. I acknowledge with gratitude the help received from Mr. Don Broach, Mrs. V. Patterson and Mr. Mike O'Neal, all of the University of Tennessee, Computer Center. Thanks also to my colleagues Mr. Mehdi Sepehri and Mr. Atul Shrivastava for their moral support.

I would like to thank my wife Anita and my son Arindam for their continuous support and countless sacrifices during the period of my graduate studies. It was God who made it all possible and I thank Him for guiding me in every step of the way.

ABSTRACT

In this dissertation research we have investigated solution approaches for the problem of scheduling of instructors (ISP) in the executive development programs offered by the Institute of Productivity through Quality at the College of Business Administration, University of Tennessee, Knoxville. There has been two principal thrusts in this research. First, we have developed and tested three different solution approaches for the Instructor Scheduling Problem (ISP). Second, we have carried out the design, development, testing, and implementation of a comprehensive decision support system (DSS) for the ISP.

In the first solution approach, the ISP has been formulated as a linear integer programming problem. Strategies based on variable/constraint elimination have been developed to reduce the problem size so that it is solvable by commercial software.

The solution time of the mathematical program exceeds forty minutes. Hence a knowledge based heuristic has been developed and implemented. The heuristic acts as an "intelligent and smart" assistant that has the initial knowledge and computing ability to find quick answers. The assistant, however, lacks the experience of the human user who must therefore guide the computer based DSS in finalizing the schedule. The knowledge based heuristic takes less than 25 seconds to generate a final schedule. Three approaches for schedule generation and four approaches for schedule improvement are investigated. This gives rise to twelve heuristics that are tested on 81 randomly generated problems and two real problems. Average Percentage Deviation

of the best heuristic from the optimal solution is found to be 1.14% for random problems and 5.93% for real problems.

In the third approach, lagrangian relaxation of ISP based on a subproblem that can be solved efficiently has been used to develop strong upper bounds and heuristic solutions. It is shown that this subproblem is an example of a special active resource scheduling problem (SARS). Three efficient solution procedures based on (a) transforming the problem to one that has a totally unimodular constraint matrix, (b) formulating the problem as a minimum cost network flow problem, and (c) formulating the problem as a transportation problem are presented for the SARS problem. Four Lagrangian Relaxations of the ISP utilizing efficient solutions of SARS as a network have been developed to yield strong upper bounds and good quality heuristic solutions. For random problems these procedures have found upper bounds that were within 100.12% of the optimal and heuristic solutions that were within 99.72% of the optimal on an average.

A hierarchical strategy for responding to the planning needs of the scheduler is proposed. This strategy suggests use of the optimal solution for generation of the first plan, monthly updating by use of the lagrangian heuristics and more frequent updating by use of the knowledge based heuristics.

The scheduling environment changes frequently due to changes in faculty availability and course offerings. Provision has been made to respond to the changes. As output, the DSS develops a final schedule, a statement of teaching assignments, a statement of work performed by instructors, and a summary statement for management control and planning.

TABLE OF CONTENTS

CHAPTER	PAGE
I. INTRODUCTION.....	1
II. LITERATURE SURVEY.....	19
III. EFFICIENT SOLUTION METHODS FOR SARS.....	30
IV. SOLUTION APPROACHES FOR INSTRUCTOR SCHEDULING PROBLEM.....	52
V. COMPUTATIONAL EXPERIMENTS.....	78
VI. IMPLEMENTATION ISSUES.....	115
VII. CONCLUSIONS AND FUTURE RESEARCH	131
BIBLIOGRAPHY.....	135
VITA.....	140

LIST OF TABLES

TABLE		PAGE
5.1	Percentage of Faculty with different Benefit Points for different Levels of Faculty Inclination.....	81
5.2	Values of Different Parameters.....	81
5.3	Range of Uniform Distributions from which Total Limits of a Faculty are Drawn	82
5.4	Strategies Used in Different Heuristics For Schedule Generation and Schedule Improvement.....	83
5.5	Comparison of 12 Heuristics on the Basis of Percentage Deviation From the Optimal Solution.....	85
5.6	Variation in Average Percentage Deviation as Number of Sessions Change.....	86
5.7	Variation in Average Percentage Deviation as Percentage of Commoncode Sessions Change.....	87
5.8	Variation in Average Percentage Deviation as Number of Faculty Changes.....	88
5.9	Variation in Average Percentage Deviation as Level of Faculty Inclination Changes.....	89
5.10	Performance of Heuristics when Benefit Values of 12, 22 and 42 are used.....	95
5.11	Performance of Heuristics when Benefit Values of 5, 15 and 35 are used.....	96
5.12	Performance of Heuristics when Benefit Values of 10, 30 and 90 are used.....	98

TABLE		PAGE
5.13	Performance of Heuristics when Benefit Values of 10, 11 and 12 are used.....	99
5.14	Performance of Heuristics when Benefit Values are sampled Randomly.....	102
5.15	Performance of the Heuristics on the Real Problems.....	104
5.16	Performance of Lagrangian Upper Bounding Procedures on Random Problems.....	107
5.17	Performance of Lagrangian Heuristics on Random Problems.....	107
5.18	Performance of Lagrangian Upper Bound Procedures on Real Problems.....	108
5.19	Performance of Lagrangian Heuristics on Real Problems.....	109

LIST OF FIGURES

FIGURE	PAGE
1.1 Hierarchical Arrangement of time periods.....	7
1.2 Constraint Matrix of an example of ISP.....	8
1.3 Optimal Non-Integral Linear Programming Solution of an ISP.....	10
1.4 The Matrix Structure of SARS with T tasks and N resources.....	17
3.1 Network Representation for an example of SARS.....	45
3.2 An Example on Use of Procedure PD.....	50
4.1 The form of $Z_D(u)$ for $m = 1$	71
5.1 Effect on Average Percentage Deviation due to Changes in Level of Faculty Inclination.....	91
5.2 Effect on Average Percentage Deviation due to Changes in the Number of Faculty	91
5.3 Effect on Average Percentage Deviation due to Changes in Number of Sessions.....	92
5.4 Effect on Average Percentage Deviation due to Changes in % of Commoncode Sessions.....	92
6.1 Final Schedule in Output File FINAL.DAT	127
6.2 Statement of Teaching Assignments.....	129
6.3 Financial Statement for Week 0821 to 0827.....	129
6.4 Summary Statistics in Data File SUMMARY.DAT	130

CHAPTER I

INTRODUCTION

The focus of this dissertation research is to investigate solution approaches for the problem of scheduling of instructors in the executive development programs offered by the Institute of Productivity through Quality at the College of Business Administration, University of Tennessee, Knoxville. There has been two principal thrusts in this research. First, we have developed and tested three different solution approaches for the Instructor Scheduling Problem (ISP). Second, we have carried out the design, development, testing, and implementation of a comprehensive decision support system (DSS) for the ISP.

In this Chapter we present a discussion on how the ISP arises and the need for a computer-based decision support system. A mathematical programming formulation of the problem is presented and some general characteristics of the problem are identified. Approaches for solving the general ISP are discussed and an easily solvable special case is identified.

1.1 PROGRAMS OF THE INSTITUTE FOR PRODUCTIVITY THROUGH QUALITY

The Institute for Productivity Through Quality (hereafter referred to as the Institute) conducts several Executive Development Programs throughout the year on an ongoing basis. Some programs are public programs in that they are open to attendance for executives of different companies. Some programs are custom designed for particular companies and are open to attendance by executives of the said companies. From a scheduling point of view, a program is a collection of sessions planned to be conducted during non overlapping time slots. The task of the scheduler is to find instructors to staff the different sessions of these programs. The problem is complicated by a number of factors.

1.1.1 Complicating Factors

There are several different factors the scheduler must consider in assigning instructors to different sessions. These factors have been outlined below:

1. Instructor preferences for teaching different programs need to be respected as far as practicable;
2. Only a subset of instructors is qualified to teach particular sessions in particular programs;
3. The number of instructors required to staff can differ across sessions;
4. For a given program, two or more similar sessions may be required to be taught by the same instructor for reasons of continuity;

5. The effectiveness of a particular program increases when particularly well developed expertise of certain instructors is utilized in sessions that they are reputed to be good at. These instructor-session combinations are to be respected;

6. The total number of assignments made to each instructor should respect prespecified lower and upper limits;

7. Instructors may not be assigned sessions that are in time conflict with their academic teaching schedules or other commitments;

8. A particular instructor may not be assigned to two sessions that are in time conflict with each other;

9. No instructor may be assigned more than three sessions on any one day ;

10. The assignment of sessions to a particular instructor should be evenly spread out over the entire semester.

11. The teaching schedules, instructor non-availabilities and session offerings rarely remain fixed even over the limited planning horizon of a semester. It should be possible to reschedule sessions without causing large numbers of changes in sessions allocated and announced to different instructors.

1.2. NEED FOR A COMPUTER BASED DECISION SUPPORT SYSTEM

At present, scheduling of instructors is done manually. This is extremely difficult and time consuming because 15 instructors are to be used to staff around 600 sessions in a typical planning period of 4 months while satisfying the many constraints. As a consequence schedules generated are far from satisfactory even though the scheduler would spend over fifty hours manually in generating the schedule. Further it

is quite difficult to make changes to the schedule in response to changes in the scheduling environment.

1.3 A MATHEMATICAL PROGRAMMING FORMULATION OF THE INSTRUCTOR SCHEDULING PROBLEM (ISP)

In this section we present a formulation of the ISP as a zero-one linear integer program.

Notation

$X_{ij} = 1$ if instructor i is assigned to session j ;

$= 0$ otherwise.

C_{ij} : "cost" of assigning instructor i to session j ;

N : total number of instructors;

T : total number of sessions;

L_i/U_i : the lower limit/upper limit on the total number
of sessions assigned to instructor i ;

P : total number of days in a planning period;

$D\{k\}$: set of sessions on day k ;

W_{im} : maximum number of sessions that may be assigned to
instructor i in week m ;

W : the total number of weeks in the planning period;

$Q\{v\}$: set of sessions in week v ;

$G\{j\}$: set of sessions in time conflict with session j ;

$H\{j\}$: set of sessions that must have the same instructor

as session j ;

A mathematical programming formulation of ISP is given below.

$$\text{Minimize } \sum_{i=1}^N \sum_{j=1}^T C_{ij} * X_{ij} \quad [1.1]$$

subject to

$$\sum_{i=1}^N X_{ij} = 1 \quad [1.2]$$

$$L_i \leq \sum_{j=1}^T X_{ij} \leq U_i ; i=1,..N; \quad [1.3]$$

$$\sum_{m \in Q\{v\}} X_{im} \leq W_{im} ; i=1,..N; v=1,..W; \quad [1.4]$$

$$\sum_{n \in D\{k\}} X_{in} \leq 3 ; i=1,..N; k=1,..P; \quad [1.5]$$

$$X_{ij} + X_{il} \leq 1 ; i=1,..N; j=1..T; l \in G\{j\} \quad [1.6]$$

$$X_{ij} - X_{ip} = 0 ; i=1,..N; j=1,..T; \text{ all } p \in H\{j\} \quad [1.7]$$

$$X_{ij} \in [0,1] , \text{ integer.} \quad [1.8]$$

The objective function [1.1] minimizes the mismatch of assigning instructor i to session j . Constraint [1.2] ensures that each session is staffed by exactly one instructor. If there is a session j that needs n instructors [$n > 1$] we replace session j with n sessions, identical to session j in all respects, except that one instructor is needed for these replacement sessions. Constraint [1.3] ensures that the total assignment to any instructor remains within prespecified lower and upper limits. Constraint [1.4] imposes

a limit on the total number of session assignments made to an instructor in a particular week. Constraint [1.5] restricts the total number of assignments made to an instructor in a day to three. Constraint [1.6] prevents assignment of an instructor to two sessions that are in time conflict. Constraint [1.7] requires that pre-specified sessions be taught by the same instructor.

1.4 SOME GENERAL CHARACTERISTICS OF THE ISP

In this section some general characteristics of the ISP are identified. Also, we show that the ISP is a special type of active resource scheduling problems. A resource is considered active if it is used to perform tasks. This is in contrast to a passive resource that is acted upon by active resources. The ISP is an active resource scheduling problem in which the demand is unary, tasks are equivalent and capacity constraints are hierarchical. The tasks of ISP are equivalent in the sense of representing equal work content. It is shown that the constraint matrix of ISP is not totally unimodular.

The ISP may also be viewed as a transportation problem where instructors are sources and sessions are the demand points. Constraint [1.2] says that demand at each destination is one unit. The important difference is that the demand must be met by supply from exactly one source. In this sense, the demand is unary.

The ISP model contains a set of capacity constraints that impose restrictions on the total number of sessions that may be assigned to an instructor during time periods of varying lengths. For example, constraint [1.3] imposes restrictions on the total number of sessions to be assigned to an instructor in a planning period which is

typically around 16 weeks. Similarly, the time periods in [1.4] and [1.5] involve one week and one day respectively.

Figure 1.1 displays the arrangement of time periods for a typical 16 week planning period. This arrangement is hierarchical in that it resembles a genetic tree. In this particular tree, children are born out of a single parent. For example, block [A] breaks into 16 non-overlapping blocks in [B]. A block in [B] breaks into 7 non-overlapping blocks in [C]. Since the time periods are hierarchical, the constraints based on them are said to be hierarchical. It may easily be seen that constraints [1.2], [1.6] and [1.7] are not hierarchical.

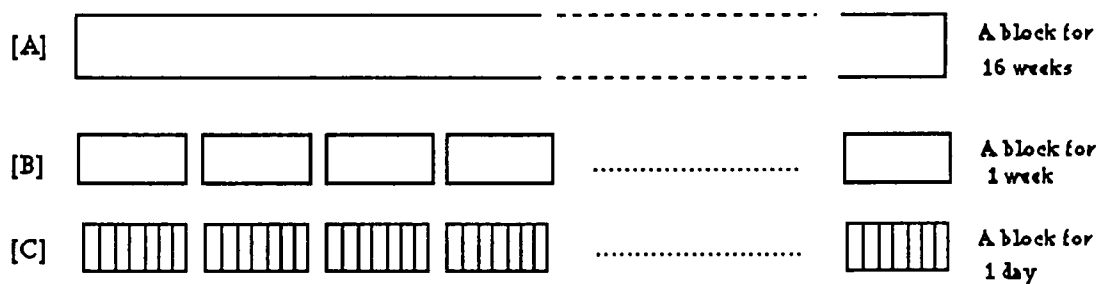


Figure 1.1 . Hierarchical Arrangement of time periods.

The hierarchical structure does not necessarily imply total unimodularity. A counter example of an ISP that is not totally unimodular is given below. First we give a precise definition of the concept of total unimodularity. An $m \times n$ integral matrix A is totally unimodular (TU) if the determinant of each square submatrix of A is equal to 0, 1 or -1 (Nemhauser and Wolsey,1988).

Consider the 2 instructor - 4 session scheduling problem presented in section 6.4. After the inequalities have been converted to equalities by use of slack/surplus, the constraint matrix is shown below (figure 1.2).

The 2 x 2 submatrix formed by the first two rows and first two columns of the constraint matrix of the example problem has a determinant of two. Hence an ISP is not TU in general.

$$\begin{bmatrix}
 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\
 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
 \end{bmatrix}
 \begin{bmatrix}
 \text{XB001} \\
 \text{XB002} \\
 \text{XB003} \\
 \text{XB004} \\
 \text{XA001} \\
 \text{XA002} \\
 \text{XA003} \\
 \text{XA004} \\
 \text{XTB} \\
 \text{XTA} \\
 \text{S1} \\
 \text{t1} \\
 \text{S2} \\
 \text{S3} \\
 \text{S4}
 \end{bmatrix}
 =
 \begin{bmatrix}
 0 \\
 4 \\
 1 \\
 0 \\
 3 \\
 1 \\
 3 \\
 1
 \end{bmatrix}$$

Figure 1.2 Constraint Matrix of an example of ISP.

Thus the constraint matrix of the Instructor Scheduling problem is not totally unimodular and so the extreme points are not guaranteed to be integers. Hence, solving the linear programming relaxation is not expected to yield the solution to the integer program. An example of a three faculty - six sessions ISP that yields a non-integral optimal solution when solved as a linear program is given in figure 1.3.

1.5 SOLUTION APPROACHES FOR THE ISP

The Instructor Scheduling Problem has been modeled as a zero-one linear integer program. Implicit enumeration approaches have been successful for integer programs (Lawler and Wood, 1966) and are often used to solve problems to optimality.

In our practical implementation the solution time for the optimal approach employing a commercial IP solver exceeded 40 minutes. Hence, knowledge based heuristics were developed to find reasonably good quality solutions in less than 30 seconds.

In order to judge the quality of heuristic solutions, lagrangian relaxation approaches are developed to find upper bounds. These approaches are extended to develop lagrangian heuristics for the ISP. In the next section we identify a subproblem of ISP that may be solved efficiently and incorporated in lagrangian relaxation approaches. This subproblem is subsequently shown to be a special case of a broader class of active resource scheduling problems [SARS]. The SARS problem is introduced and discussed in the subsequent section. In Chapter IV we discuss four lagrangian relaxations of the ISP that utilize efficient solution methods for the SARS problem to develop strong upper bounds and good heuristic solutions for ISP.

[a] Mathematical Programming Formulation

Maximize

$$10X_{11} + 10X_{12} + 10X_{13} + 10X_{14} + 10X_{15} + 10X_{16} \\ + 10X_{21} + 10X_{22} + 10X_{23} + 10X_{24} + 40X_{25} + 40X_{26} \\ + 40X_{31} + 40X_{32} + 40X_{33} + 40X_{34} + 40X_{35} + 40X_{36}$$

subject to

$$\begin{aligned} X_{11} + X_{21} + X_{31} &= 1 \\ X_{12} + X_{22} + X_{32} &= 1 \\ X_{13} + X_{23} + X_{33} &= 1 & \text{[number of faculty required]} \\ X_{14} + X_{24} + X_{34} &= 1 \\ X_{15} + X_{25} + X_{35} &= 1 \\ X_{16} + X_{26} + X_{36} &= 1 \end{aligned}$$

$$\begin{aligned} 2 \leq X_{11} + X_{12} + X_{13} + X_{14} + X_{15} + X_{16} &\leq 3 \\ 1 \leq X_{21} + X_{22} + X_{23} + X_{24} + X_{25} + X_{26} &\leq 3 & \text{[limits]} \\ 1 \leq X_{31} + X_{32} + X_{33} + X_{34} + X_{35} + X_{36} &\leq 4 \end{aligned}$$

$$\begin{aligned} X_{11} - X_{12} &= 0 \\ X_{21} - X_{22} &= 0 \\ X_{31} - X_{32} &= 0 & \text{[common code constraints]} \\ X_{13} - X_{14} &= 0 \\ X_{23} - X_{24} &= 0 \\ X_{33} - X_{34} &= 0 \end{aligned}$$

$$\begin{aligned} X_{11} + X_{15} &\leq 1 \\ X_{21} + X_{25} &\leq 1 & \text{[time conflict constraints]} \\ X_{31} + X_{35} &\leq 1 \\ X_{13} + X_{16} &\leq 1 \\ X_{23} + X_{26} &\leq 1 \\ X_{33} + X_{36} &\leq 1 \end{aligned}$$

$$0 \leq X_{ij} \leq 1; i=1,..3; j=1,..6.$$

[b] Optimal Solution

Objective value = 180;

$$\begin{aligned} X_{11} &= 0.5; X_{31} = 0.5; X_{12} = 0.5; X_{32} = 0.5; \\ X_{33} &= 1.0; X_{34} = 1.0; X_{25} = 1.0; X_{16} = 1.0; \end{aligned}$$

Figure 1.3 Optimal Non-Integral Linear Programming Solution of an ISP.

1.6 A SUBPROBLEM OF ISP

In this section we discuss a subproblem of the Instructor Scheduling Problem (SISP) for which efficient solution methods may be developed.

The following two assumptions are necessary to reduce ISP to SISP :

- a) Constraints [1.6] are not present, i.e. there are no sessions that are in time conflict.
- b) Constraints [1.7] are not present, i.e. there are no blocks of sessions that must be taught by a common instructor.

Constraints [1.1] through [1.5] and [1.8] constitute the problem SISP. Approaches for solving SISP efficiently are discussed below.

SISP is a zero-one linear integer program. It can be solved efficiently by transforming it to a problem with a totally unimodular constraint matrix that can be solved as a linear program and yet be guaranteed of an integral solution. Alternatively, SISP can be formulated as a network flow problem or a transportation problem and solved efficiently.

This transformation allows an analyst who has access to either a linear programming code, a minimum cost network flow code or a transportation code to solve SISP or problems similar to SISP. Efficient solution approaches for this subproblem are utilized to develop lagrangian relaxation approaches for the ISP.

1.7 THE SARS PROBLEM

In the previous section we introduced a subproblem of the Instructor Scheduling Problem (SISP) that can be solved efficiently. SISP has three distinct sets of constraints: a) a set of assignment constraints, b) a set of hierarchical structure capacity constraints, and c) constraints enforcing integrality of variables. It is this structure that makes it possible for us to solve the problem efficiently. However, SISP is not the only problem with this structure. In fact a broad class of zero-one linear integer programming problems have this structure and consequently can be solved efficiently. This special class of active resource scheduling problem would be referred to as SARS. The problem SARS is of practical importance because it has wide applicability. In this section we define the terminology, present a general formulation of the SARS model along with the matrix structure in a general case.

1.7.1 Terminology

In the following discussion it is assumed that all variables appear on the left hand side of a constraint. Two constraints are said to be parallel if their left hand sides are a scalar multiple of one another. We assume that redundant parallel constraints have been eliminated and every constraint has at most one parallel constraint.

Let $\{B\}$ be the set of constraints for a given system. Assume that all constraints in $\{B\}$ have 0, 1 left hand side coefficients.

Definition:

The cardinality of a constraint is defined to be the number of variables with non-zero left hand side coefficients.

Definition:

For any constraint $C \in \{B\}$, consider all constraints $\{A\} \subseteq \{B\}$ whose variables with non-zero coefficients are a strict superset of the variables with non-zero coefficients in C. The constraint(s) in $\{A\}$ with minimal cardinality is (are) called the parent(s) of C. If P is a parent of C, C is a child of P.

Definition:

A constraint set is said to be a Hierarchical Structure Constraint Set [HSCS] or is said to satisfy the hierarchical structure if its constraints satisfy all of the following:

(i) Every constraint has at most one parent constraint if parallel constraints are counted as only one constraint;

(ii) Any two children of the same parent constraint are either parallel or consist of mutually exclusive subsets of variables with non-zero coefficients in the parent constraint.

(iii) The variables with non-zero coefficients in every constraint are a subset of the variables with non-zero coefficients in the constraint with the largest cardinality.

The constraint with the largest cardinality in (iii) above is called the root constraint.

example:

Consider the following constraint set:

$$x_{10} + x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} \leq 4 \quad [1.9]$$

$$x_{10} + x_{11} + x_{12} + x_{13} + x_{14} \leq 3 \quad [1.10]$$

$$x_{10} + x_{11} + x_{12} \leq 2 \quad [1.11]$$

$$x_{13} + x_{14} \leq 1 \quad [1.12]$$

It may be verified that this constraint set is HSCS.

Definition:

A set of HSCS constraints is said to be perfect if for every parent constraint P, the union of distinct variables with non-zero coefficients in its children yields the variables with non-zero coefficients in P.

example 1:

$$x_1 + x_2 + x_3 + x_4 + x_5 \leq 3 \quad [1.13]$$

$$x_1 + x_2 \leq 1 \quad [1.14]$$

$$x_3 + x_4 + x_5 \leq 2 \quad [1.15]$$

This HSCS constraint set is perfect.

example 2 :

$$x_1 + x_2 + x_3 + x_4 + x_5 \leq 3 \quad [1.16]$$

$$x_1 + x_2 \leq 1 \quad [1.17]$$

$$x_4 + x_5 \leq 1 \quad [1.18]$$

This HSCS constraint set is not perfect.

In further discussions we assume that a given HSCS constraint set is perfect. If it is not perfect we look at each parent constraint P. If the union of distinct variables with non-zero coefficients of the children does not yield the variables with non-zero coefficients of P, identify the missing variables with non-zero coefficients in set {H}. Add a redundant " $\leq$ " constraint whose left hand side is expressed as a sum of elements in {H} and right hand side is set equal to the sum of upper bounds of elements in {H}. This is repeated for every parent so that the HSCS constraint set becomes perfect.

A perfect HSCS will be referred to as PHSCS.

1.7.2 A General Formulation of SARS

In this subsection a mathematical programming formulation of SARS is presented along with an example of the matrix structure of SARS.

Let:

$X_{ij} = \{ \begin{array}{l} 1 \text{ if active resource } i \text{ is assigned to task } j \\ 0 \text{ otherwise } \end{array} \};$

C_{ij} = Cost of assigning active resource i to task j ;

N = Total Number of active resources;

T = Total Number of tasks;

The general formulation of SARS is shown below:

$$\text{Minimize } \sum_{i=1}^N \sum_{j=1}^T C_{ij} * X_{ij} \quad [1.19]$$

Subject to :

$$\sum_{i=1}^N X_{ij} = 1 \quad ; j=1, \dots, T \quad [1.20]$$

$$PHSCS_i \quad ; i=1, \dots, N \quad [1.21]$$

$$X_{ij} \in [0,1], \text{ integer} \quad [1.22]$$

The objective function [1.19] minimizes the cost of assigning active resource i to task k . Constraint [1.20] ensures that each task is staffed by exactly one active resource. Constraint [1.21] imposes limits on the total number of tasks that can be assigned to an active resource. For each resource there is one set of PHSCS capacity constraints. The

root constraint of $PHSCS_i$ contains only the zero-one variables pertaining to resource i . Further the root constraint of $PHSCS_i$ contains all the variables pertaining to resource i .

The matrix structure of SARS for N resources and T tasks is shown in figure 1.4. In figure 1.4, RHS stands for the right hand side and $PHSCS_i$ stands for the perfect HSCS pertaining to resource i .

In this subsection we introduced the problem SARS which has the same characteristics as that of the problem SISP. As a result it is possible to develop efficient solution methods for SARS (Chapter III). These efficient solutions would then be incorporated in lagrangian relaxation techniques to develop upper bounds and heuristic solutions to the ISP. These approaches are discussed in Chapter IV.

1.8 APPROACHES FOR A DECISION SUPPORT SYSTEM

A major emphasis, in this dissertation research, has been on development and implementation of a user friendly decision support system (DSS). We have analyzed the requirements of the scheduler and designed and developed a comprehensive DSS to support decision making of the scheduler. The DSS accepts data in natural format, solves the problem, generates the final schedule in natural format and generates required statements.

A hierarchical strategy is proposed to meet the planning needs of the user. This strategy includes use of an optimal approach for generating the first schedule, use of lagrangian heuristics for monthly updating of schedule and use of knowledge based heuristics for adjustments to the current month's schedule.

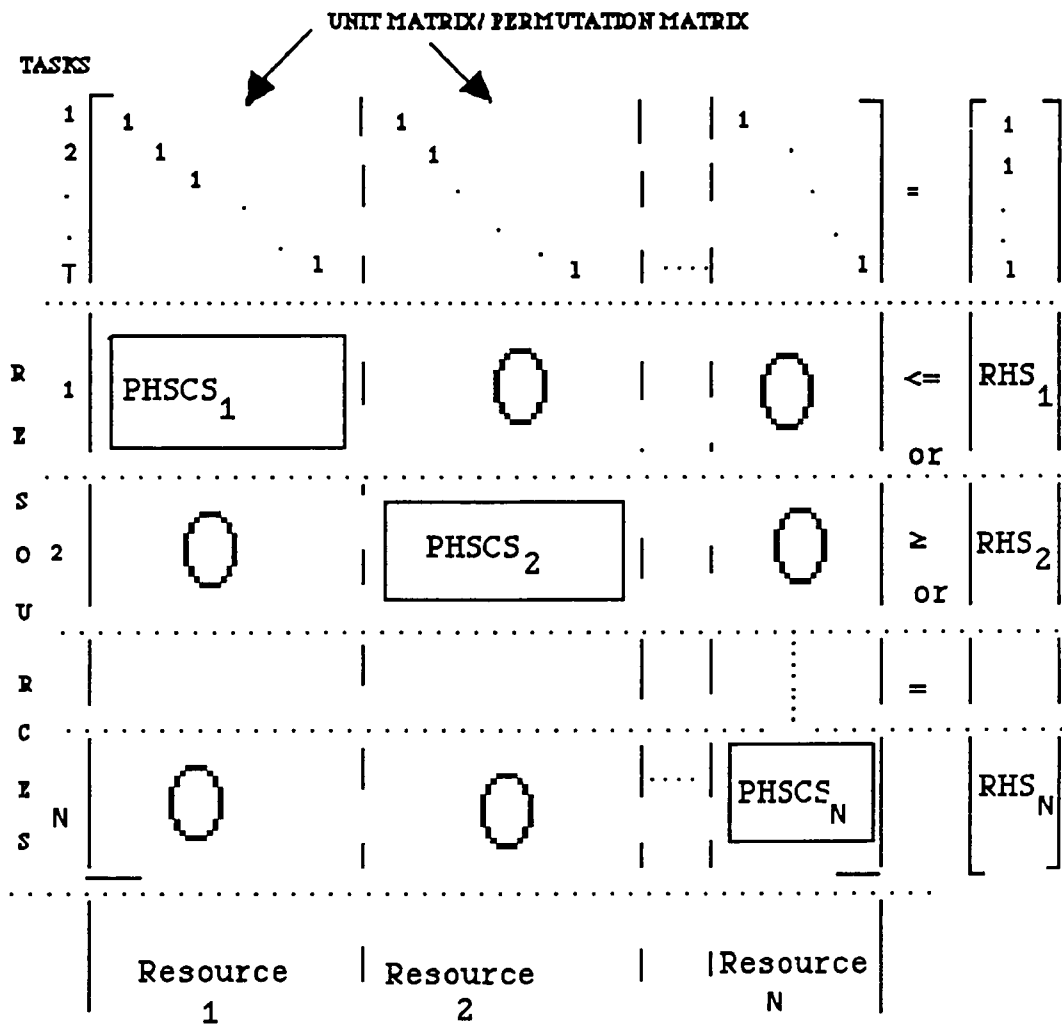


Figure 1.4 The matrix structure of SARS with T tasks and N resources.

1.9 OVERVIEW OF CHAPTERS

Chapter II discusses the relevant literature. In Chapter III, three efficient solution methods for the SARS problem are presented. In Chapter IV we present an optimal approach, a knowledge based heuristic approach and a lagrangian relaxation approach for solving ISP. Chapter V discusses the computational results. Implementation issues relating to problem generation, use of LINDO and data processing are presented in Chapter VI. A few promising research topics are identified in Chapter VII.

CHAPTER II

LITERATURE SURVEY

In this chapter the relevant literature has been surveyed. This consists of research into the literature pertaining to problems similar to SARS, personnel scheduling problems and decision support systems.

2.1 PROBLEMS SIMILAR TO SARS

Problems similar to SARS include the Assignment problem, Bounded Interval Generalized Assignment problem and the Capacitated Transportation problem.

2.1.1 The Assignment Problem [AP]

The Assignment Problem arises when we need to staff N positions by N persons. Let $X_{ij} = 1$ if person i is assigned to position j ; otherwise $X_{ij} = 0$. C_{ij} is the cost of assigning person i to position j . The mathematical programming formulation of the AP is:

$$\text{Minimize } \sum_{i=1}^N \sum_{j=1}^N C_{ij} * X_{ij} \quad [2.1]$$

subject to

$$\sum_{i=1}^N X_{ij} = 1 \quad ; (j=1,..N); \quad [2.2]$$

$$\sum_{j=1}^N X_{ij} = 1 \quad ; (i=1,..N); \quad [2.3]$$

$$X_{ij} \in [0,1] , \text{ integer} ; (\text{ all } i,j) \quad [2.4]$$

This problem is an example of SARS where [2.3] is the only constraint of the perfectly hierarchical structure constraint set (PHSCS).

The regular AP has also been considered as a special case of a transportation problem, a minimum weight matching problem and a minimum cost network flow problem.

The AP can be reduced to a minimum cost flow problem and solved with exactly N flow augmentations. Each flow augmentation can be completed by a shortest path computation of $O(N^2)$ complexity. Thus the AP is solvable in polynomial time of $O(N^3)$ (Lawler, 1976). For specific techniques developed to solve the AP see (a) Kuhn (1955) for the Hungarian method, (b) Barr, Glover and Klingman (1977) for the alternating path basis algorithm, (c) Hung (1983) for a polynomial simplex method and (d) Balinski (1985) for signature methods.

2.1.2 The Bounded Interval Generalized Assignment Problem [BIGAM]

This problem arises when N persons are to be used to staff T positions with the added condition that a person may be assigned to more than one position. For each person i , r_{ij} is the resource used up if person i is assigned to position j . The total resource used for each person i must lie within lower limit a_i and upper limit b_i . Ross, Soland and Zoltners (1980) introduce BIGAM which is formulated as follows:

$$\text{Minimize } \sum_{i=1}^N \sum_{j=1}^T C_{ij} * X_{ij} \quad [2.5]$$

subject to

$$\sum_{i=1}^N X_{ij} = 1 \quad ; (j=1,..T); \quad [2.6]$$

$$a_i \leq \sum_{j=1}^T r_{ij} * X_{ij} \leq b_i \quad ; (i=1,..N); \quad [2.7]$$

$$X_{ij} \in [0,1] , \text{ integer}; (\text{ all } i,j); \quad [2.8]$$

BIGAM is a generalization of a problem called the Generalized Assignment Problem (Ross and Soland, 1975) where $a_i = 0$ for all i .

BIGAM is of practical interest because it is a case where multiple tasks can be assigned to each person as long as the agent resource used up by performing the assigned tasks fall within a specified interval. BIGAM and GAP are solved by branch and bound incorporating lagrangian relaxation techniques (Ross, Soland and Zoltners, 1980).

Our interest lies in the special case when $r_{ij} = 1$ for all i,j . This special case has been referred to as the Semi-Assignment Problem (Ross, Soland and Zoltners, 1980).

2.1.2.1 The Bounded Interval Semi-Assignment Problem [BISAM]

The BISAM is stated as follows:

$$\text{Minimize} \quad \sum_{i=1}^N \sum_{j=1}^T C_{ij} * X_{ij} \quad [2.9]$$

subject to

$$\sum_{i=1}^N X_{ij} = 1 \quad ; (j=1,..T); \quad [2.10]$$

$$a_i \leq \sum_{j=1}^T X_{ij} \leq b_i \quad ; (i=1,..N); \quad [2.11]$$

$$X_{ij} \in [0,1] , \text{ integer}; (\text{ all } i,j); \quad [2.12]$$

To impose the lower limit $\sum_{j=1}^T X_{ij} \geq a_i$ for all i , one may introduce an upper bounded slack variable S_i (with upper bound of $b_i - a_i$) and re-write [2.11] and [2.12] as [2.13] and [2.14].

$$\sum_{j=1}^T X_{ij} + S_i = b_i ; (i=1,..N); \quad [2.13]$$

$$X_{ij} \in [0,1], \text{ integer} ; (\text{all } i,j); 0 \leq S_i \leq (b_i - a_i) \text{ all } i; \quad [2.14]$$

Ross, Soland and Zoltners (1980) suggest that [2.9], [2.10], [2.13] and [2.14] be solved as a linear program by the simplex method since the constraint matrix is totally unimodular.

However, BISAM is an example of SARS. [2.11] constitutes a set of twin root constraints that are the only set of constraints in PHSCS. Thus it has a significantly more efficient network flow solution as shown in Chapter III.

2.1.3 Capacitated Transportation Problem [CTP]

Transportation models with upper bounds on flows between origins and destinations are referred to as CTP. The CTP is stated as follows:

$$\text{Minimize} \quad \sum_{i=1}^N \sum_{j=1}^T C_{ij} * X_{ij} \quad [2.15]$$

subject to

$$\sum_{i=1}^N X_{ij} = b_j ; (j=1,..T) \quad [2.16]$$

$$\sum_{j=1}^T X_{ij} = a_i ; (i=1,..N) \quad [2.17]$$

$$X_{ij} \leq U_{ij} ; (\text{all } i,j) \quad [2.18]$$

$$X_{ij} \geq 0; (\text{all } i,j) \quad [2.19]$$

The a_i and b_j are positive integers corresponding to the supply and demand at source i and demand point j . X_{ij} is the shipment between these points. U_{ij} is an upper bound on X_{ij} and represents a capacity restriction on the flow in arc (i,j) . We assume that $\sum_{i=1}^N a_i = \sum_{j=1}^T b_j$.

Charnes and Cooper (1957) and Ford and Fulkerson (1957) have considered the above problem and provided algorithms based on the transportation tableau to solve the problem. Our interest, however, is in the special case when $U_{ij}=1$ for all i,j . This special case is an example of SARS since the demand point j with demand of b_j may be replaced by b_j number of demand points each with a demand of 1 unit.

2.2 PERSONNEL SCHEDULING PROBLEM

The Instructor Scheduling Problem as described in Chapter I has not been explicitly reported in the literature. However, there are many instances where researchers have formulated and solved scheduling/assignment problems of similar nature. Scheduling problems that have similarities to the Instructor Scheduling Problem can be classified into two groups:

1. personnel scheduling in public systems ;
2. scheduling in academic settings .

2.2.1 Personnel Scheduling in public systems.

Personnel scheduling in public systems has received extensive attention from the operations research community. In particular, scheduling of nursing personnel in hospitals has been studied in great depth.

2.2.1.1 Scheduling of Nursing Personnel.

Maier-Rothe and Wolfe (1973) reported on the successful development and implementation of a formal system for the scheduling and allocation of nursing staff. Their approach is based on their ability to generate feasible patterns of working days and off-days on a heuristic basis. Generated alternative cyclical patterns are considered by the nursing staff and the choice of the particular pattern to be worked is made by the nurses. The advantage is that the choice is left to the nurses rather than imposing a decision on them.

Abernathy, Baloff, Hershey and Wandel (1973) formulated the nurse scheduling problem as a stochastic programming problem. They suggested an iterative solution procedure using random loss functions and permitted the inclusion of statistically dependant demands.

Warner (1976) formulated the nurse-scheduling problem as a large multiple-choice programming problem whose objective function quantifies preferences of individual nursing personnel concerning length of work-stretch, rotation patterns and requests for days off. The constraints provide for minimum numbers of nursing personnel of each skill class to be assigned to each day and shift of four or six-week scheduling period. The problem is solved by a modification of Balintfy and Blackburn's algorithm for multiple-choice programming problems. A description of the implementation of the system in sixteen units of the University of Michigan Hospital and units of Duke University Hospital is provided.

Miller, Pierskalla and Rath (1976) formulated the nurse scheduling problem as one of selecting a configuration of nurse schedules that minimize an objective function that balances the tradeoff between staffing coverage and schedule preferences of individual nurses, subject to feasibility constraints on the nurse schedules. The

problem is solved using a cyclic coordinate descent algorithm that finds near optimal solutions.

Megeath (1978) reported on development and implementation of a "quick-and-dirty " method for scheduling of nursing personnel. The procedure described was manual and did not involve the use of computers.

2.2.1.2 Scheduling a Patient Transportation Service.

A patient transportation service is an ancillary service organized for the purpose of moving patients from one location to another, as the need arises, during their stay in hospital. Magazine (1977) reported on using standard queueing analysis to determine the minimum manpower needed at different times to meet the demand (within five minutes of its arising) 95 % of the time. Then based on these requirements, a mixed integer linear program is solved to find the optimal schedule. The objective of the program is to minimize the total size of the work force subject to ensuring that minimum number of personnel are available and the personnel get a meal break in the allowable range.

2.2.1.3 Scheduling of Telephone Operators

Segal (1974) described the formulation of scheduling of telephone operators as a minimum cost flow problem that is solved by use of the out-of-kilter algorithm (Fulkerson, 1961). They incorporate a post-processor to allow the user to override the suggested schedule.

2.2.1.4 Scheduling Police Patrol Cars

Kolesar, Rider, Crabill and Walker (1975) reported on using queueing theory and integer linear programming for scheduling patrol cars so that specified service standards are met at each hour of the day. Queueing theory is used to estimate patrol-car requirements at different times of the day. An integer linear program is used to find the number of patrol cars that start patrol-duty at the beginning of each hour and the times at which they go out of service for meals. MIP subroutines of MPSX are used to solve the integer linear program. The model is reported to be under test by the New York City Police Department.

2.2.1.5 Personnel Assignment in the Navy

The U.S. Navy has been trying for years to develop a model to automate its personnel assignment process. The requirement of rotating navy personnel frequently between sea duty and shore duty in accordance with multiple policies makes modeling efforts difficult. However, without a model to optimize assignments with regard to multiple policies, navy managers do not know the real impact of personnel assignments. Liang and Thompson (1987) reported on formulating this problem as a capacitated transshipment problem which was then solved by using a commercially available computer code, namely GNET.

2.2.2 Scheduling in Academic Settings

Hill, Naumann, and Chervany (1983) described two successfully implemented systems for personnel assignment. The first problem concerned assignment of students to half-hour slots for company interviews. The problem is formulated as a generalized personnel assignment model with the objective of

maximizing utility. This problem was solved by using the network optimization code GNET. The second problem concerned assignment of students to class projects and was solved by use of the out-of-kilter algorithm. The authors attributed the success of the two systems to (a) finding a solution that satisfied the constraints, (b) collecting data with a view to assessing assignee preferences and (c) developing a comprehensive data processing system that provided an interface among data sources, the solution algorithm and the users.

McClure and Wells (1984) presented an integer programming model for assignment of courses to faculty members. Unlike previously proposed models, the decision variables represent the assignment of complete teaching schedules rather than courses to faculty members. Faculty members were provided with a set of feasible schedules and were asked to assign a utility value to each schedule. The objective function was oriented towards maximizing the sum of all faculty members' utilities for their assigned schedules subject to constraints of staffing all sections and assigning one schedule to a faculty member. The models were solved using the commercial software package LINDO.

Plotnicki and Garfinkel (1986) addressed the problem of scheduling courses to quarters (or semesters) with the goal of minimizing the total time students require to graduate, while maintaining a feasible schedule with regard to staffing. Complicating factors influenced the authors to develop an interactive simulation model which captured most of the complexities of the general problem. The model was tested with data from the MBA program at the University of Tennessee, Knoxville.

Glassey and Mizrach (1986) developed a large zero-one integer program as part of a decision support system for assigning classes to rooms at the University of California at Berkeley. The model involved half a million variables and around 27000

constraints. Because of its large size, exact optimal solutions were not attempted. Instead, a very fast heuristic was used to generate schedules of reasonable quality.

Ardekani and Mahmood (1986) considered the problem of assigning students to groups for class projects. They used a heuristic approach and an optimization approach (an assignment model) to assign both experienced and inexperienced students to different project groups with the objective of creating a balanced and fair environment.

2.3 DECISION SUPPORT SYSTEMS

Researchers have reported about design and implementation of decision support systems (DSS) in the operations research literature regularly for over ten years. The intention behind surveying this literature was to identify principles laid down by previous researchers and learn about costly pitfalls that need to be avoided in designing a DSS. Another objective was to identify recent trends in design of user friendly systems.

Bell (1985) pointed out that visual interactive models gained the confidence of decision makers because the latter could personally validate the models.

Sullivan and Secrest (1985) described a simple optimization-based DSS for production planning at Dairyman's Cooperative Creamery Association (DCCA). The DCCA is the largest single milk processing plant at one location in the U.S. An interactive and user friendly primal linear programming model coded in BASIC was used to determine the daily production and inventory levels so as to maximize plant throughput.

Goslar, Green and Hughes (1986) defined a decision support system as a flexible computer-based system that can assist managers in decision making when confronted with a semi-structured or unstructured problem. The authors pointed out that a DSS allows a decision maker to focus on problem solving without technical, clerical and computational distractions.

Ow and Smith (1987) described two design principles for knowledge-based systems and described how these principles were used to develop a knowledge-based job shop scheduling system. A knowledge based system or expert system is one that emulates the decision making of human experts. The study used LISP and SRL as the programming languages and reported scheduling 460 operations in the job shop by using about seven hours on the VAX 11/780. It was recognized that program execution in the LISP/SRL environment was extremely slow compared to, for example, FORTRAN.

Henderson (1987) pointed out the need for key stakeholders to participate in the design and implementation of a DSS. The author formally put on record the widespread feature of DSS practice wherein the model builder is only a "chauffeur", that is, the model builder's role is to support and facilitate the decision maker in the building of the DSS.

CHAPTER III

EFFICIENT SOLUTION METHODS FOR SARS

In this chapter three efficient solution methods for SARS are discussed. These methods are utilized in a lagrangian relaxation approach to develop upper bounds and heuristics for the ISP (Chapter IV).

The SARS model is a zero-one linear integer program. If the linear program (L.P.) solution of SARS were integral we would not have to resort to implicit enumeration approaches like branch and bound to find an integral solution. A L.P. solution is guaranteed to be integral if the constraint matrix is totally unimodular. But, we have not been able to show that the SARS problem has a totally unimodular constraint matrix. Thus there is no guarantee that L.P. solutions would be integral. However, a procedure to transform the SARS model to one that has a totally unimodular matrix has been developed and is described. The modified SARS problem can then be solved as a linear program. Because of total unimodularity of the constraint matrix, an integer solution is guaranteed. Network and transportation algorithms are typically a hundred or more times faster than linear programming methods. Thus it is advantageous to solve a problem by a network/transportation algorithm. Procedures that transform SARS into a minimum cost network flow problem or transportation problem are described. Thus an analyst who has access to any one of a linear programming code, a minimum cost network flow code or a transportation code will be able to solve SARS efficiently.

3.1 PROCEDURE TO CONVERT SARS INTO A PROBLEM WITH A TOTALLY UNIMODULAR MATRIX

In this section we present Procedure PB that transforms SARS into a formulation that has a totally unimodular constraint matrix. The advantage is that the problem may now be solved as a linear program and yet integral solutions are guaranteed.

The basic idea of the procedure is given below. We look at a Perfect Hierarchy Structure Constraint Set (PHSCS) for each resource. The idea is to work with parent constraints in the order of increasing cardinality, starting with the parent constraint of smallest cardinality. Each child of the parent is grouped into a new variable. If it is a " $\leq$ " constraint (" $\geq$ " constraint), the new grouped variable has an upper bound (lower bound) equal to the right hand side of the constraint. In the next appearance of this child the newly defined variable is used in place of the child. In this manner new variables are defined in a way so that eventually every variable appears once with a +1 coefficient and once with a -1 coefficient.

In the following discussion, X_k are the original variables in the problem. Let V_i be the new variables that are defined procedurally as sums of subsets of X_k . Let N be the total number of resources and T be the total number of tasks. We denote a child constraint by CC and its parallel, if any, by CCP. The right hand sides of CC and CCP would be denoted by RHS1 and RHS2 respectively. Let RO1 and RO2 be the relational operators of CC and CCP respectively. Let J be a counter that keeps track of the new variables that are defined.

3.1.1 Procedure PB

Step 0 : Initialization. Set $I = 1$; Set $J = 0$; $R = \{\emptyset\}$;

Step 1 : Select resource I for variable definition; {Step 1 through step 4 will be carried out for the PHSCS of each resource.}

Step 2 : From PHSCS_I , identify every constraint that is a parent constraint and include it in the LIST. Sort the LIST in increasing order of cardinality;

{ We are creating a list of parent constraints. After sorting, this list has the smallest constraint at the top. }

Step 3 :

a) Select one constraint from top of LIST for variable grouping;
[PC constraint];

{ The parent with smallest cardinality is selected first; A new grouped variable will be created for each child of PC. }

b) For PC, select its child constraints and put in set Q ;

{ The children of PC are identified as set Q };

c) From set Q select one constraint [CC] and its parallel [CCP], if any.

{ This particular CC and CCP will be grouped as a new variable };

Set $J = J+1$; { increment the counter for new variables };

d) Include the variables of CC in set P .

{ The specific variables of this child are identified and included in P };

e) Search P for subsets of variables X_k whose sum has already been defined as a new variable V_i . { If the sum of a particular set of variables has already been defined as a new variable V_i , then in subsequent appearances, the sum of this particular set of variables should be referred to as V_i }. Among these V_i , find BCV, that is V_i with biggest cardinality and include it in set S . { If a particular set of variables appear

in more than one V_i then we need to consider only the V_i with largest cardinality}. Delete elements of BCV from P. Repeat until all variables that have been grouped within some V_i are deleted from P. { As a consequence, set P would now contain all the variables X_k that have not been included in any V_i . }

f] Replace CC and CCP by the new constraint [3.1]:

$$V_J - \sum_{X_k \in P} X_k - \sum_{V_i \in S} V_i = 0 \quad [3.1]$$

{As a result of this step the two constraints CC and CCP are replaced by one new constraint. This new constraint has the effect of defining a new grouped variable as a sum of minimal number of variables. Moreover, as a consequence of this step, a "-1" appears in the column of original X_k variables and predefined group variables V_i . At the end of this step a "+1" appears in the column of the new group variable V_J defined in this step. }

g] Define bounds on variable V_J :

$$L_J \leq V_J \leq U_J$$

where U_J and L_J are defined as follows:

$$\begin{aligned} U_J &= \text{RHS1 if RO1 is " } \leq \text{ " ; or} \\ &= \text{RHS2 if RO2 is " } \leq \text{ " ; or} \\ &= |P| + \sum_{V_i \in S} U_i \text{ otherwise;} \end{aligned}$$

where U_i is an upper bound on V_i .

$$\begin{aligned} L_J &= \text{RHS1 if RO1 is " } \geq \text{ " ; or} \\ &= \text{RHS2 if RO2 is " } \geq \text{ " ; or} \\ &= \sum_{V_i \in S} L_i \text{ otherwise} \end{aligned}$$

where L_i is a lower bound on V_i .

{ The new grouped variable defined above replaces CC and CCP. These constraints are capacity constraints imposing lower and upper limits on sums of subsets of variables. Hence the new group variable V_J needs to be bounded by appropriate lower and upper limits. }

h) Delete CC and CCP if any from Q;

Set $P = \emptyset$; Set $S = \emptyset$;

i) If $Q = \emptyset$, go to step 3j, otherwise go to step 3c;

j) If PC is not the root constraint, delete PC and its parallel if any from LIST and go to step 3k; If PC is the root constraint, include it and its parallel if any in set Q, repeat steps 3c through 3h, set $LIST = \{ \emptyset \}$ and go to step 3k.

k) If $LIST = \emptyset$, set $R = \{ R \cup V_J \}$;

go to step 4;

otherwise go to step 3a; { If all the children of a particular parent PC have been grouped, we need to identify another parent to work on. }

{ We need to keep track of the new grouped variable that represents the root constraint. The identity of this root constraint is stored in set R. }

Step 4 : Set $I = I + 1$; If $I > N$, go to step 5; otherwise go to step 1;

Step 5 : Define a redundant constraint [3.2] involving all the group variables of root constraints.

$$- \sum_{V_m \in R} V_m = -T \quad [3.2]$$

{ This is the last step in the algorithm. It defines a redundant constraint that has the effect of putting a "-1" in the columns of new group variables that represent root constraints. }

3.1.2 Lemma 3.1

When procedure PB is applied to an instance of problem SARS, the resulting problem has a constraint matrix that is totally unimodular.

Proof:

The constraint matrix that results after procedure PB is applied to SARS will be referred to as MSARS to stand for modified SARS. The node-arc-incidence-matrix of a network has exactly two non-zeros in each column, one of which is a +1 and the other is a - 1. We will refer to this property as NAIM. We will work towards showing that MSARS satisfies the NAIM property. The nomenclature of Procedure PB is carried over to this discussion.

The matrix structure of SARS has been shown in figure 1.4. However we present a simplified representation of the matrix structure to help in understanding of the following logic.

Sub-segment 1	P	P	P	P	P

Subsegment 2	PHSCS ₁	O	O	O	O
	O	PHSCS ₂	O	O	O
	O	O	PHSCS ₃	O	O
	O	O	O	PHSCS ₄	O
					O
	O	O	O	O	PHSCS _n
	A	B	C	D	

P stands for a permutation matrix. A permutation matrix is a unit matrix with zero or more columns permuted. $PHSCS_i$ stands for perfect hierarchy structure constraint set involving variables of resource i only. The matrix represented as O is a zero matrix of appropriate dimension. A, B, C, D and E stand for column segments. Each column segment is independent of any other column segment in the sense that each column segment involves variables of only one resource. Hence if we could show that procedure PB acting on column segment A converts it into a form whose columns satisfy the NAIM property then the same reasoning would be valid for the other segments too. Thus we would look only at the columns in segment A.

There are two cases that arise.

Case 1: $PHSCS_1$ consists of a root constraint and one or more children of the root constraint.

Segment A has two non-zero sub-segments. Sub-segment 1 consists of Matrix P which has a +1 in every column of segment A (one such in each row of sub-segment 1). Sub-segment 2 contains $PHSCS_1$ and several O blocks. $PHSCS_1$ consists of variables pertaining to resource 1 all arranged in constraints that obey the perfect hierarchy structure.

When procedure PB is applied to $PHSCS_1$, every child constraint CC and its parallel, if any, is replaced by one constraint involving a new grouped variable. The procedure PB works its way through the task of grouping variables in ascending order of cardinality of a constraint. The first time that a variable set is grouped, step 3f

ensures that the original variables X_k appear with a -1 coefficient. At this point the new grouped variable appears with a +1 coefficient. Any variable that has once been grouped under a group variable V_i will thereafter never appear as an original variable. Thus the original variables X_k appear once with a +1 coefficient in P and then with a -1 coefficient in modified $PHSCS_i$.

As the procedure progresses, a particular sub-group of variables in CC reappears only in a constraint with larger cardinality than CC . Suppose that this particular sub-group of variables had been aggregated under some V_i in step 3f. In a subsequent iteration this particular sub-group is referred to as V_i and appears with a -1 coefficient in step 3f. This particular V_i loses its distinct identity and is aggregated under some V_k . Thus the variable V_i appears once with a +1 coefficient and again with a -1 coefficient.

Since the constraint set is $PHSCS$, every original variable X_k is grouped under some group variable V_i before procedure PB reaches the root constraint. When the root constraint is grouped in step 3f of some iteration, a variable V_r is defined in terms of other V_i 's. At this point all V_i 's, other than V_r have appeared once with a +1 coefficient and then again with a -1 coefficient.

The only column with a single entry is that of V_r . It has a +1 in this column. However, the identity of V_r is stored in set R at step 3k. At the end of investigation of all resources, a redundant constraint is defined in step 5 of procedure PB where the variables stored in set R (namely the V_r 's) appear with a -1 coefficient.

Thus at the conclusion of procedure PB , every variable, new or old, appears twice. Once it appears with a +1 coefficient and then again with a -1 coefficient.

Case 2: PHSCS₁ consists of the root constraint only.

In this case, step 3f of procedure PB yields a constraint of the form:

$$V_r - \sum_{X_k \in \text{Root Constraint}} X_k = 0$$

Thus each original variable X_k appears once with a +1 coefficient and then again with a -1 coefficient. V_r appears with a +1 coefficient at this stage. However, in step 5 of procedure PB, V_r appears with a -1 coefficient.

In view of the above discussion it is seen that each column in the modified matrix has exactly two non-zeros, a +1 and a -1. In other words MSARS satisfies the NAIM property. Further we know that a matrix that satisfies the NAIM property is totally unimodular. Hence we conclude that MSARS is totally unimodular. This completes our argument.

3.1.3 An Example on Using Procedure PB :

Consider the following example of problem SARS :

X_{11}	$+X_{21}$	$= 1$
X_{12}	$+X_{22}$	$= 1$
X_{13}	$+X_{23}$	$= 1$
X_{14}	$+X_{24}$	$= 1$
X_{15}	$+X_{25}$	$= 1$
X_{16}	$+X_{26}$	$= 1$
$X_{11}+X_{12}+X_{13}+X_{14}+X_{15}+X_{16}$		≤ 4
$X_{11}+X_{12}+X_{13}+X_{14}+X_{15}+X_{16}$		≥ 2
$X_{11}+X_{12}+X_{13}+X_{14}$		≤ 3
	$X_{15}+X_{16}$	≤ 1

$$X_{21}+X_{22}+X_{23}+X_{24}+X_{25}+X_{26} \leq 5$$

$$X_{21}+X_{22}+X_{23}+X_{24}+X_{25}+X_{26} \geq 2$$

$$X_{21}+X_{22} \leq 1$$

$$X_{23}+X_{24}+X_{25}+X_{26} \leq 4 \quad [a]$$

$$X_{ij} \in [0,1], \text{ Integer}$$

[SEGMENT A] [SEGMENT B]

Note that constraint [a] is the redundant constraint added to make the constraints in segment B of perfect hierarchical structure. If Procedure PB is applied to the above example of SARS, we get the following MSARS:

X_{11}	$+X_{21}$	$= 1$
X_{12}	$+X_{22}$	$= 1$
X_{13}	$+X_{23}$	$= 1$
X_{14}	$+X_{24}$	$= 1$
X_{15}	$+X_{25}$	$= 1$
X_{16}	$+X_{26}$	$= 1$
$V_1 - X_{15} - X_{16}$		$= 0$
$V_2 - X_{11} - X_{12} - X_{13} - X_{14}$		$= 0$
$-V_1 - V_2 + V_3$		$= 0$
$V_4 - X_{21} - X_{22}$		$= 0$
$V_5 - X_{23} - X_{24} - X_{25} - X_{26}$		$= 0$
$-V_4 - V_5 + V_6$		$= 0$
$-V_3 - V_6$		$= -6$

$$X_{ij} \in [0,1] , \text{ Integer}$$

$$0 \leq V_1 \leq 1; \quad 0 \leq V_2 \leq 3; \quad 2 \leq V_3 \leq 4; \quad 0 \leq V_4 \leq 1; \quad 0 \leq V_5 \leq 4; \quad 2 \leq V_6 \leq 5;$$

3.2 PROCEDURE TO CONVERT SARS TO A MINIMUM COST NETWORK FLOW PROBLEM

Network algorithms are 100 or more times faster than linear programming codes on problems that can be formulated in both ways (Klingman, 1976). Thus it would always be advantageous to seek a network representation of a linear program as this would give us significant computational advantages. In this section we present a procedure that converts SARS into a network problem. Any algorithm that finds a minimum cost flow through a capacitated network may be used to efficiently solve SARS.

The basic idea is described below. A source s and sink t are created with a supply and demand equal to the total number of tasks. A node R_I is set up for each resource I . A directed arc is drawn from s to R_I with lower/upper bounds on flow equal to the lower/upper limits on total assignments to a resource. This takes care of the root constraint for each resource. The root constraint is now used to expand the network (current parent). Consider the set of distinct children of the current parent. For each distinct child DC , a node is created and a directed arc is drawn from the corresponding current parent node to the child node DC . Lower and Upper bounds on flow for this arc are determined on the basis of restrictions imposed by the particular constraint DC and its parallel if any. For example, the right hand side of a " $\geq$ " constraint (" $\leq$ " constraint) becomes a lower bound (upper bound) on flow in this arc.

This is repeated for all constraints working down consistently from parents to its children. When all constraints are exhausted, identify the nodes that do not have emanating arcs (FC nodes). Set up a node for each task (task node). Each of these FC nodes represent a constraint involving a unique subset of variables. Consider a particular FC node. The corresponding variables are assignment variables of a particular resource to particular tasks. Identify the set of tasks $\{H\}$ in this subset of variables. Draw a directed arc from the FC node to each node that represents a task in $\{H\}$. When these arcs have been drawn for all FC nodes, draw a directed arc from each task node to the sink. This arc will have an upper bound of one unit on the flow in this arc. An FC node represents a constraint involving variables of only one particular resource. Suppose that this resource is l . Consider one task node m that is in $\{H\}$ of this FC node. Assign a cost to the arc from l to m equal to C_{lm} where C_{lm} is the cost of assigning resource l to task m . This is done for all FC nodes. A cost of zero is assigned to all the other arcs.

The notation used in the procedure is given below. Let (i,j) be a directed arc from node i to node j . An arc will always mean a directed arc. The designations $[lb,ub,cost]$ over or under an arc would represent the lower bound and upper bound on flow in the arc and cost of one unit of flow in that arc. Let C be the current child constraint and let $RHS1$ be the right hand side of C . If C has a parallel constraint, let $RHS2$ be the right hand side of the parallel of C . Let P denote the parent node of current child constraint C and let $PHSCS_i$ represent the perfect hierarchy structure constraint set for resource i . The relational operator of C is denoted RO with N being the total number of resources and T as the total number of tasks. Let C_{ik} be the cost of assigning resource i to task k .

3.2.1 Procedure PC :

Step 0: Set $I = 1$; Set $J = 1$;

Step 1: Set up a source node s with a supply of T units. Set up a sink node t with a demand of $-T$ units.

Step 2: Select resource I for network definition. Include every constraint of $PHSCS_I$ in the LIST. Sort the LIST in decreasing order of cardinality.

Step 3: Select one constraint from top of LIST for network expansion and refer to this constraint as C .

IF $J = 1$:

- a] Set $P = s$; {The source is the parent constraint}
- b] Set up a node R_I ;
- c] Draw an arc from s to R_I ;
- d] Set parameters $[lb, ub, cost]$ by calling sub-procedure Set-Parm;
- e] Include in set U_I the tasks for which variables are included in constraint C ;

IF $J > 1$:

- a] Set $P =$ Node that represents the parent constraint of C ;
- b] Set up a node IA_J ;
- c] Draw an arc from P to IA_J ;
- d] Set parameters $[lb, ub, cost]$ by calling sub-procedure Set-Parm;
- e] For each node IA_J include in set V_{IJ} the tasks for which variables are included in constraint C ;

Step 4: a] Set $J = J+1$; Delete C and its parallel if any from the LIST.

b] IF $LIST = \emptyset$, Go to Step 5;

c] IF $LIST \neq \emptyset$, Go to Step 3;

Step 5: a] Set $I = I + 1$;

- b] Set $J = 1$;
- c] IF $I > N$ Go to Step 6;
- d] IF $I \leq N$ Go to Step 2;

Step 6:

- a] Set up a node N_k for each task k ; ($k = 1, \dots, T$);
- b] Draw an arc (N_k, t) from each N_k to t ;
- c] For this arc (N_k, t) set:

$$[lb, ub, cost] = [0, 1, 0]$$

- Step 7:
- a] For all nodes IA_J and R_I that have no emanating arc, draw an arc to node N_k if task k is included in V_{IJ} or U_I ;
 - b] For this particular arc, set:
- $$[lb, ub, cost] = [0, 1, C_{Ik}]$$

3.2.2 Sub-Procedure Set-Param:

This Sub-Procedure is called upon to set the parameters of a particular arc .

IF C does not have a parallel :

IF RO of C is " $\geq$ " :

$$lb = RHS1; ub = M; cost = 0;$$

IF RO of C is " $\leq$ " :

$$lb = 0; ub = RHS1; cost = 0;$$

IF C has a parallel constraint :

IF RO of C is " $\geq$ " :

$$lb = RHS1; ub = RHS2; cost = 0;$$

IF RO of C is " $\leq$ " :

$$lb = RHS2; ub = RHS1; cost = 0;$$

Note: M is a very large number; (e.g. $M = T$).

3.2.3 An Example of Using Procedure PC :

We consider the same example as considered for procedure PB in section 3.1.3. When procedure PC is applied to the above example problem we get the network in figure 3.1.

3.3 PROCEDURE TO CONVERT SARS TO AN UNCAPACITATED TRANSPORTATION PROBLEM

In this section we present a procedure that produces a standard uncapacitated transportation tableau representation of SARS. This version of SARS may now be solved very efficiently by any transportation code that an analyst may have access to. A procedure developed by Wagner (1959) to convert a capacitated transportation problem with certain types of capacity constraints into an enlarged uncapacitated transportation problem has been adapted to develop procedure PD.

The basic idea of the procedure is discussed below. A transportation tableau is set up with tasks as destinations and resources as sources. The tableau is balanced so that total demand is equal to total supply by introducing a dummy task. If a " $\geq$ " constraint does not have a parallel " $\leq$ " constraint, a redundant " $\leq$ " constraint is introduced in the system.

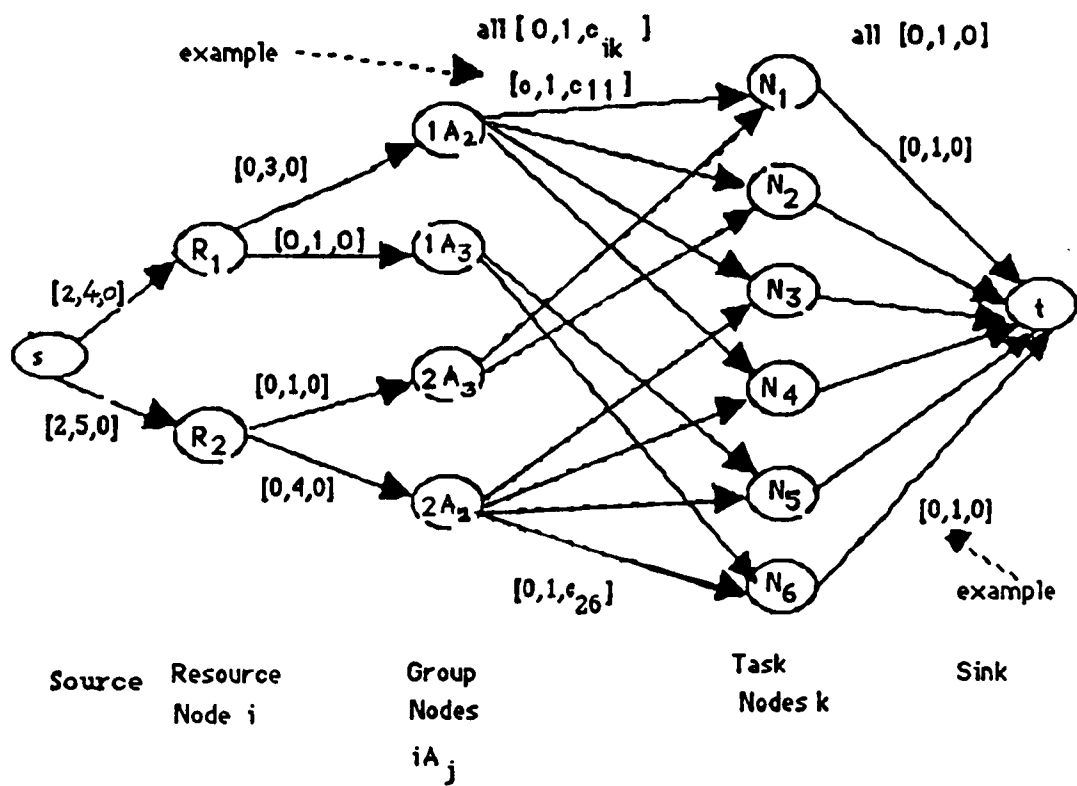


Figure 3.1 Network representation for an example of SARs.

For each of the " $\leq$ " root constraints, a new source and a new destination are introduced. The demand of the new destination is equal to the right hand side of the " $\leq$ " root constraint reduced by the right hand side of the " $\geq$ " root constraint if any. The supply at the new source is equal to the right hand side of the " $\leq$ " root constraint. If it is a root constraint of resource I, the supply of row I in the tableau is reduced by the right hand side of the " $\geq$ " root constraint if any. For each parent C in the constraint set a new destination is set up with a total demand equal to the sum of the right hand sides of its " $\leq$ " children reduced by the sum of the right hand sides of its " $\geq$ " children. For each " $\leq$ " child C_C of C a source is set up with an availability equal to the right hand side of C_C . The total availability of the row corresponding to C is reduced by the right hand side of " $\geq$ " parallel of C_C if any. Appropriate large costs M are assigned so that use of certain cells in the tableau are prohibited.

The notation used in the procedure is given below. Let C_{ij} be the cost of assigning resource i to task j and d_{ij} be the cost of sending one unit from source i to destination j. Let J be a counter for the number of sources and K be a counter for the number of destinations excluding the dummy destination. Let A_i be the availability at source i and D_j be the demand at destination j. Let N be the total number of resources and T be the total number of tasks. The cardinality of a constraint is denoted by $|C|$. Let RHS be the right hand side of a constraint. In any given pair of parallel constraints C, the " $\leq$ " constraint will be referred to as $C(L)$ while the " $\geq$ " constraint, if any, will be referred to as $C(G)$. If any " $\geq$ " constraint $C(G)$ does not have a parallel, add a redundant " $\leq$ " constraint $C(L)$ such that $C(L)$ is parallel to $C(G)$ and has a RHS equal to $|C(G)|$. Thus every " $\geq$ " constraint has a parallel " $\leq$ " constraint but not vice-versa. A " $\leq$ " constraint that does not have a parallel is also referred to as $C(L)$.

3.3.1 Procedure PD

The steps of the procedure are given below.

Step 0: Set $I=1$; $J=N$; $K=T$; $LIST=\{ \emptyset \}$;

Step 1: a] Set up a transportation tableau with N sources (corresponding to the N resources) each with an availability of T units and T destinations (corresponding to the T tasks) each with a demand of one unit.

b] Set up a dummy destination 0 with a demand equal to $(N-1)(T)$. Verify that total supply = total demand = NT .

c] Assign costs: $d_{ij} = C_{ij}$; $(i=1,...N; j=1,...T)$;

$d_{i0} = 0$; $(i=1,...N)$;

Step 2:

a] Select resource I for tableau expansion;

b] Include every parent constraint of $PHSCS_I$ in $LIST$;

c] Sort the $LIST$ in decreasing order of cardinality;

Step 3:

a] Select a constraint and its parallel if any from top of $LIST$. This set of parallel constraints is to be used in tableau expansion (PTE). $PTE(L)$ is the L constraint in the set PTE . $PTE(G)$ is the G constraint in PTE .

b] IF $PTE(L)$ is not a root constraint go to Step 5.

Step 4:

a] Set $J=J+1$; $K=K+1$; $Q=I$;

b] Add a new source J and a new destination K to the tableau. J is the row number in tableau for PTE .

c] IF BOTH $PTE(L)$ AND $PTE(G)$ EXISTS

$A_J = \text{RHS of } PTE(L)$;

$$A_Q = (A_Q) - (\text{RHS of PTE}(G));$$

$$D_K = (A_J) - (\text{RHS of PTE}(G));$$

d] IF ONLY PTE(L) EXISTS

$$A_J = \text{RHS of PTE}(L);$$

$$D_K = (A_J);$$

e] Include in set U the destinations for which variables are included in PTE(L);

f] Call sub-procedure MODCOST to set new costs and modify costs;

Step 5: Expand tableau for each child of PTE(L):

a] Let CD be the set of all children of PTE(L);

b] Let Q be the row number in tableau corresponding to PTE(L);

c] Set $K=K+1$; Add a new destination K to the tableau;

$$d] \text{ Set } D_K = \sum_{H \in CD} (\text{RHS of } H(L)) - \sum_{H \in CD} (\text{RHS of } H(G));$$

e] Select a child $H \in CD$; set $J=J+1$; Add a new source J;

This J is the row number in the tableau for this H;

f] Set $A_J = \text{RHS of } H(L)$;

g] Include in set U the destinations for which variables are included in H;
set costs by calling sub-procedure MODCOST;

h] IF H(G) EXISTS:

$$\text{Set } A_Q = A_Q - \text{RHS of } H(G);$$

i] Delete H from CD;

j] Repeat [e] through [i] of step 5 until CD is empty;

Step 6:

Eliminate PTE(L) and PTE(G) if any from LIST;

Step 7:

IF LIST = { $\emptyset$ } go to step 8; otherwise go to step 3;

Step 8:

IF I = N, STOP;

IF I < N, Set I = I+1; go to step 2.

3.3.2 Sub-procedure MODCOST

This procedure is used repeatedly to modify costs in the tableau.

Set or Modify costs of tableau as follows:

a] $d_{QK} = d_{JK} = 0$;

b] $d_{Jl} = d_{Ql}$ for all $l \in U$;

c] $d_{Ql} = M$ for all $l \in U$;

d] $d_{Jl} = M$ for all $l \notin U$ and $l \neq K$;

e] $d_{pK} = M$ for all rows P that have an unassigned cell in
column K;

M is a large cost that would prohibit assignment to the cell.

3.3.3 An Example on using procedure PD:

We consider the same example as that considered for procedure PB in section 3.1.3. When procedure PD is applied to the above example problem, we get the tableau given in figure 3.2.

DESTINATIONS												
Source	0	1	2	3	4	5	6	7	8	9	10	A _j
1	0	M	M	M	M	M	M	0	M	M	M	4
2	0	M	M	M	M	M	M	M	M	0	M	4
3	M	M	M	M	M	M	M	0	0	M	M	4
4	M	C11	C12	C13	C14	M	M	M	0	M	M	3
5	M	M	M	M	M	C15	C16	M	0	M	M	1
6	M	M	M	M	M	M	M	M	M	0	0	5
7	M	C21	C22	M	M	M	M	M	M	M	0	1
8	M	M	M	C23	C24	C25	C26	M	M	M	0	4
D _K	6	1	1	1	1	1	1	2	4	3	5	

Note: D_K is Demand at destination K; A_j is the availability at source j; C_{ij} stands for the cost of assigning resource i to task j; M is a large cost.

Figure 3.2 An Example on Use of Procedure PD

3.4 GENERAL OBSERVATIONS

In this Chapter, three methods for solving SARS efficiently have been presented. While we suspect that solving SARS as a linear program would yield integral solutions, we could not establish this to be the case. Instead, we presented a procedure that transformed SARS into a problem with a totally unimodular constraint matrix. This modified problem can be solved as an L.P. and be guaranteed of an integral solution. Further, methods are presented to transform SARS into (a) a minimum cost network flow problem and (b) a balanced uncapacitated transportation problem. Existing algorithms may be used to solve these transformed problems efficiently.

In Chapter IV we utilize the network solution of SARS to develop lagrangian relaxation approaches for finding upper bounds and heuristic solutions to ISP.

CHAPTER IV

SOLUTION APPROACHES FOR INSTRUCTOR SCHEDULING PROBLEM

In this chapter we describe three solution approaches for ISP. In the first section we describe strategies that are pursued in order to find an optimal solution using a commercial linear integer programming software. The time taken to find an optimal solution exceeds 40 minutes. Knowledge based heuristics that obtain reasonably good quality solutions in less than 25 seconds are described in the second section. Lagrangian relaxation approaches to compute upper bounds and find heuristic solutions are described in the third section.

4.1 OPTIMAL APPROACHES USING COMMERCIAL SOFTWARE

In this section we discuss the mathematical programming formulation of the Instructor Scheduling problem (ISP) as presented in section 1.3 with the objective of identifying strategies and methods of problem solving that could be incorporated in a robust decision support system (DSS). Efficient solution strategies applicable to the SARS problem are not directly applicable to ISP as the necessary assumptions are not valid. Hence we discuss solution strategies that would be useful in conjunction with a general purpose linear integer programming software.

The mathematical programming formulation has the advantage of being comprehensive. However, for the real problems studied, it gives rise to a program with 7658 zero-one variables and over 28300 constraints.

One strategy is to reduce the size of the problem by eliminating variables that must take on zero value in the final solution and by eliminating redundant constraints. Elimination of variables is justified because such a step may reduce the solution time. Elimination of constraints in an integer programming problem is usually beneficial if it does not lead to an increase in the duality gap. These issues are discussed in this section.

4.1.1 Discussion on Problem Formulation

The ISP has been formulated in section 1.3 as a zero-one integer linear program with "cost" minimization as the objective. The "cost" C_{ij} is a measure of the degree of mismatch between instructor i and session j . However, it is easier for the user to work with a measure for the degree of match between a faculty and a session than degree of mismatch. The degree of match between the kind of faculty needed for a session and the background of an instructor may be measured in terms of "benefit". In particular, if instructor i is assigned to session j a "benefit" or "utility" of B_{ij} units accrues to the Institute. Hence, in developing a DSS, we treat ISP as a maximization problem with the objective of maximizing the total "benefit" that accrues to the Institute.

A particularly significant characteristic of the formulation is its comprehensiveness. The formulation is an attempt to formalize mathematically the most serious and repeatedly articulated concerns of the actual scheduler.

The formulation produces a non-trivial zero-one integer program with NS zero-one integer variables where N is the number of instructors and S is the number of sessions. For a typical planning period with 550 sessions and fifteen available instructors this corresponds to 8250 zero-one integer variables. The number of constraints actually observed in the planning period of Fall 1988 is above

28000. Thus this formulation turns out to be a challenging one for present day commercially available software.

4.1.2 Reduction of Problem Size

Strategies for variable elimination and elimination of redundant constraints are appropriate for this problem and are pursued. Brearley and Mitra (1975) reported that many advantages result if a mathematical programming problem is reduced in dimension to an equivalent problem before submitting it to an algorithm for solution. In particular, the time taken to solve a linear programming (L.P.) problem is generally reduced if the problem is made smaller. For an L.P. the solution time increases approximately proportionally with the number of variables and approximately with the square of the number of constraints (Schrage, 1981). Another advantage is that core storage requirements are reduced if the problem is made smaller. Since the commercially available software for solving integer programming problems are based on a branch and bound scheme that require repeated computation of solutions of linear programs associated with nodes of the branch and bound tree, a smaller problem is solved in less time. However, there are some exceptions to this claim and we will discuss them later in this sub-section.

4.1.2.1 Variable Elimination

It is advantageous to reduce the number of zero-one variables by eliminating the variables that are required to take on zero values in the solution. This reduces the number of variables that may take on fractional values in a linear programming solution thus reducing the total time in getting to the optimal. This strategy has been followed in

generation of the integer program and is found to yield significant savings as shown in Chapter VI.

4.1.2.2 Constraint Elimination

In order to reduce the size of the problem, another strategy would be to eliminate redundant constraints. The concept of "redundant" constraint is discussed. For I.P. problems elimination of constraints may actually lead to an increase in solution times. This happens because the associated L.P. formulation may become weaker after elimination of constraints. The concept of the strength of a formulation is discussed. Examples of pitfalls in constraint reduction are presented.

4.1.2.2.1 Redundant Constraints

In this sub-section we provide definitions of "redundant" constraints and "wholly redundant constraints". We would be interested in reducing the size of the problem by elimination of redundant constraints if this step does not cause an increase in the duality gap. It is observed that elimination of "wholly redundant constraints" does not increase the duality gap and hence may be eliminated.

We consider the problem in the following form.

Maximize cx
 subject to $Ax \leq b$ Problem P
 $x \geq 0$, integer.

The L.P. relaxation of problem P is $\bar{P}$:

Maximize cx
 subject to $Ax \leq b$ Problem $\bar{P}$
 $x \geq 0$.

We assume that P and $\bar{P}$ are feasible. We make this assumption because we have no interest in developing strategies for an infeasible problem.

Definitions

- a) $V(A)$ is the optimal objective value of problem A .
- b) $F(A)$ is the feasible region of problem A .
- c) The duality gap is the difference between the optimal objective value of the L.P. relaxation of an I.P. problem and the optimal objective value of the I.P. problem. To be specific, $\text{duality gap} = V(\bar{P}) - V(P)$

Definitions

- a) A constraint of an I.P. problem is said to be redundant if removal of the constraint has no effect on the feasible region $F(P)$ or the value of $V(P)$.
- b) A redundant constraint of an I.P. problem is said to be wholly redundant if removal of the constraint has no effect on the feasible region $F(\bar{P})$ or the value of $V(\bar{P})$.

From the definition of a "wholly redundant" constraint it follows that its elimination will not increase the duality gap. However, if you eliminate a "redundant" constraint there is no guarantee that the duality gap would not be increased.

4.1.2.2.2 Strength of Formulation

The concept of the strength of formulation of an integer program has been discussed by many researchers. To be very specific, if the linear programming (L.P.) relaxation of an integer programming (I.P.) formulation A of a problem yields a stronger bound than the L.P. relaxation of I.P. formulation B of the same problem,

formulation A is considered to be better than formulation B when a L.P. based solution procedure (e.g. LINDO or MPSX/MIP) is used. This concept is better understood through an example adapted from Nemhauser and Wolsey (1988).

Consider the following integer program .

[F1] Formulation 1 : $\text{Max } 4A + 3B + 2C + D$

subject to $93A + 49B + 37C + 29D \leq 111$

$A, B, C, D \in [0, 1], \text{ integer}$

It may be verified that the only feasible solutions to the above constraint are:

[AAA] $[0\ 0\ 0\ 0]; [1\ 0\ 0\ 0]; [0\ 1\ 0\ 0]; [0\ 0\ 1\ 1];$

$[0\ 0\ 1\ 0]; [0\ 0\ 0\ 1]; [0\ 1\ 1\ 0]; [0\ 1\ 0\ 1].$

Consider two other ways of formulating the above problem.

[F2] $\text{Max } 4A + 3B + 2C + D$

subject to $2A + B + C + D \leq 2$

$A, B, C, D \in [0, 1], \text{ integer}$

[F3] $\text{Max } 4A + 3B + 2C + D$

subject to $2A + B + C + D \leq 2$

$A + B \leq 1$

$A + C \leq 1$

$A + D \leq 1$

$A, B, C, D \in [0, 1], \text{ integer}$

It is easily seen that the constraints of all the three formulations are satisfied by the solutions in [AAA] and that these are the only $[0, 1]$ solutions for these

constraints. In other words all the three formulations correctly describe the 4 dimensional space for which [AAA] are the [0,1] solutions.

The L.P. relaxations and the I.P. version of all the three formulations were solved using LINDO. The results are summarized below.

Formulation Number	L.P. solution	I.P. solution	Duality Gap
F1	6.79	5.00	1.79
F2	6.00	5.00	1.00
F3	5.00	5.00	0.00

Thus it is seen that the third formulation F3 has the smallest duality gap. In other words, F3 is the strongest formulation although it contains three redundant constraints.

The above example illustrates how one formulation may be stronger than another in the sense of yielding a L.P. solution that is close to the I.P. objective value.

In a solution procedure based on branch and bound, the L.P. solutions of the sub-problems are used as bounds to curtail the search to nodes that have better bounds than the incumbent. The incumbent is the best integer solution found so far. If the bounds are poor, i.e. the duality gap is large, many nodes in the tree may be explicitly examined. This happens because the bounds of the nodes appear promising although in reality they may not have a better integer solution than the incumbent. If the bounds are strong, i.e. the duality gap is small, a larger number of nodes are likely to be implicitly enumerated as more of the nodes would be worse as compared to the incumbent. Since

fewer nodes are likely to be explicitly examined, the total solution time is likely to be less.

In the above discussion we have given an example to show how the presence of redundant constraints may actually lead to a reduction in solution times. We now present an example where elimination of redundant constraints does not increase the duality gap and hence does not increase the solution time.

example:

Maximize $X + Y$

subject to $2X + Y \leq 1.5$ [4.1]

$1.33X + Y \leq 1$ [4.2]

X, Y integer and $\in [0, 1]$.

It may be verified that $(0, 0)$ and $(0, 1)$ are the only feasible integer solutions to the above problem. The optimal I.P. solution is $(0, 1)$ with an objective value of 1. The optimal L.P. solution is $(0, 1)$ with an objective value of 1. In this case the duality gap is zero.

If you eliminate constraint [4.2], it may be verified that $(0, 0)$ and $(0, 1)$ are the only feasible integer solutions to the reduced integer problem. Hence Constraint [4.2] is redundant to the integer problem. The optimal integer solution of the reduced I.P. is $(0, 1)$ with an objective value of 1. The optimal solution of the L.P. relaxation of the reduced I.P. is also $(0, 1)$ with an objective value equal to 1. Thus the duality gap is zero. Thus [4.2] is a wholly redundant constraint.

So far we have presented two examples. In the first example we showed a case where the duality gap increases if redundant constraints are eliminated. The second example shows that there may be cases where elimination of redundant constraints will not have any effect on the duality gap.

4.2 KNOWLEDGE BASED HEURISTIC APPROACH

In this section we give reasons why a heuristic was considered necessary and point out how the heuristic approach fits into the overall decision support system. A unique push-pull Knowledge Based Heuristic is proposed for the instructor scheduling problem. We highlight the salient characteristics that distinguish this heuristic from other scheduling heuristics reported in the literature. The heuristic is in the class of intelligent systems that act in a way similar to human beings. In contrast to the philosophy of "expert systems", our proposed system was developed with the perspective of keeping decision making in the hands of the decision maker. It is this feature that has gone a long way towards making the DSS acceptable to the user. Acceptability of the DSS depends in part on its ability to respond to changes in the scheduling environment. Methods used by the heuristic to respond to changes have been recorded. The user also has the option of carrying out manual adjustments to the schedule.

4.2.1 Rationale for a Heuristic

The primary reason for developing the heuristic was the need of the user for generating a schedule in a short time. It took us about 48 minutes of CPU time to generate an optimal schedule by use of the mathematical program. The user found this to be too long. Further there was no guarantee of a feasible solution in the very first run. Hence the user may have to go through a few iterations to get a feasible solution. Thus there was a need for generating a good schedule in a short period of time.

4.2.2 Approach taken in Heuristic Development

The approach taken by us differs from "expert" systems which are being developed by many researchers. There are two reasons for not building "expert" systems using artificial intelligence languages like PROLOG or LISP. First, full scale "expert" systems take a long computation time (Ow and Smith, 1987). Second, "expert" systems take decision making away from the decision maker (Ow and Smith, 1987). The system advocated by us takes shorter computation time as fewer rules are built in. Further, it acts as an intelligent assistant of the decision maker. Rules are used to carry out the detailed exercise of finding a preliminary schedule. The human scheduler is then prompted to use his experience and judgement in guiding the program towards a solution acceptable to him. Thus as a matter of principle, decision making has been left in the hands of the decision maker. This particular feature had a major influence in the acceptance of the heuristic by the actual scheduler. In our opinion, systems built with this approach of providing intelligent assistance to the decision maker are likely to have greater acceptance at the hands of users.

Let us use an analogy. The system that we have designed operates more like a junior official in an organization. Generally, in an organization, a decision making problem gets defined and acted upon in a file. A junior official is entrusted with the file and is expected to come up with preliminary recommendations. Usually this stage of the task requires a great deal of planning, meticulous data collection and large amounts of computation. At the end of this stage the problem is defined quantitatively and a few attractive alternatives are identified based on a set of guidelines. After the junior official completes the allotted task, the recommendations on the file are sent for review to a higher official. The higher official uses years of accumulated experience to review the recommendations in the file and suggests directions for improving the

output. The file iterates between the two officials until an acceptable solution is found. In this process, experience of the higher official is used to make a seasoned decision.

In an analogous manner, our system acts like a junior official and constructs a schedule on the basis of prespecified rules. These rules are of two types. The first type of rules specify the order in which sessions are to be scheduled. For example, sessions that need the same set of three instructors are scheduled first. The second type of rules specify how to select an instructor for a particular session. For example, an instructor with highest benefit for a session is always considered first. After the schedule is constructed, the work is presented to the user for review. Thereafter, the user draws on past experience to issue directions to the system for improving the proposed solution. The process iterates until the user is satisfied.

4.2.3 Knowledge Based Heuristic

The heuristic consists of four phases. The first phase consists of generating a schedule that is feasible except for the fact that it may not respect the constraints that limit the total number of sessions assigned to an instructor between the upper and lower limits specified by the said instructor. In the second phase the heuristic identifies the instructors for whom these constraints have been violated and tries to make adjustments so that total allocations to all instructors are brought within the limits requested. In the third phase KBH uses a built in strategy to improve the schedule with the objective of finding one of better objective function value. In the fourth phase KBH presents a summary of the results of its efforts and seeks the guidance of the user regarding directions for further processing.

The division of the work into these four phases has been made to imitate the actual way the user proceeds manually for development of the schedule.

4.2.3.1 Strategies for Schedule Generation

Three strategies are investigated for generation of the schedule:

a) Strategy A: This strategy tries to keep uniformity with regard to the proportion assigned. The strategy selects the faculty with smallest proportion assigned; breaks ties in favor of faculty with higher OFC; (OFC = objective function coefficient). The "proportion assigned" is the proportion of "goal" of a faculty that has been achieved at any point in schedule generation. The "goal" of a faculty is the mid-point between upper and lower limits on total assignments that can be made to a faculty.

b) Strategy B: This strategy always chooses faculty with higher OFC; breaks ties in favor of faculty with smaller proportion assigned.

c) Strategy C: In this strategy, the faculty with highest Local Composite Score (LCS) is chosen for each session. The LCS is computed as shown below.

$$\text{LCS} = \text{Local composite score} = \text{OFC}[I,J] * [K(I)];$$

The values of $K(I)$ depends on proportion assigned (PA) as shown below.

<u>Range of Proportion Assigned</u>	<u>$K(I)$</u>
0 < PA ≤ 0.2	5
0.2 < PA ≤ 0.4	4
0.4 < PA ≤ 0.6	3
0.6 < PA ≤ 0.8	2
0.8 < PA ≤ 1.0	1

$K(I)$ tries to drive the heuristic towards selecting a faculty with lower proportion assigned. LCS tries to drive the heuristic towards selecting a faculty with highest product of OFC and $K(I)$.

4.2.3.2 Strategies for Schedule Improvement

Assuming a feasible solution has been found, four strategies are investigated for improvement of the schedule.

Let J be a session under consideration. For session J , let I be the faculty currently assigned to it and $\{P\}$ be the set of potentially assignable faculty with better OFC than I . Among the faculty in $\{P\}$, let B be the faculty with the best OFC.

a) Strategy 1 : 2 Faculty Replacement (2FR)

This strategy aims at finding a faculty with better OFC to replace the currently assigned faculty. The basic steps of the strategy are described below.

For each session J , identify I , set $\{P\}$ and B . Replace I by B . Repeat for all sessions.

b) Strategy 2 : 2 Faculty Interchange (2FIC)

In this strategy assignments of two faculty are interchanged if it results in a net increase in objective value. The basic steps of the strategy are described below.

For each session J , identify I , set $\{P\}$ and B . Among sessions assigned to B find one where I can be assigned. Carry out the interchange if it results in a net increase in objective value. Repeat for all sessions.

c) Strategy 3 : 3 Faculty Interchange (3FIC)

In this strategy assignments of three faculty are interchanged if such an interchange results in a net increase in objective value. The basic steps of the strategy are described below.

For each session J , identify I , set $\{P\}$ and B . Identify a third faculty K to act as a facilitator of interchange. This faculty K is selected randomly. Among sessions assigned to K find one where I can be assigned. Among sessions assigned to B find

one where K can be assigned. Carry out the interchange if it results in a net increase in objective value. Repeat for all sessions.

d) Strategy 4 : Combination Heuristic (CH)

Execute improvement strategy 1, 2 and 3 one after another in this sequence.

For each strategy for schedule generation there are four strategies for schedule improvement. In other words, there are a total of 12 heuristics that we have investigated.

4.2.3.3 User Guidance in Schedule Refinement

In the fourth phase KBH presents a menu to the user for controlling directions of further processing. The user has the options of (a) looking at the current assignments, (b) increasing assignments of a particular faculty, (c) decreasing assignments of a particular faculty, (d) reallocating sessions from one specified faculty to another specified faculty, (e) changing the upper and lower limits on total assignments and finding a revised schedule under the new scenario and (f) printing the solution. In a typical session the user starts by studying the current solution. Then the other menu options are used to guide KBH in refining the schedule.

4.2.3.4 Performance of KBH

For a typical planning period of around 500 sessions KBH finds a schedule in around 25 seconds. Because of its speed there is clearly a bias in the users' mind in favor of KBH as compared to the exact optimization based DSS.

4.2.4 Responding to Changes

Success of the DSS depends on its ability to respond satisfactorily to changes in the scheduling environment. The DSS has been equipped to respond to all possible changes identified by the user. A Fortran program generates the revised schedule for the new scenario. The user has the option of guiding the computer program to improve the schedule.

Changes in faculty data and structural changes are the two kinds of changes that are possible. Changes in faculty data include changes in faculty availability, faculty inclination, faculty teaching schedule and limits on total assignments. Structural changes change the structure of the problem. Structural changes include adding/deleting a program offering, adding/deleting a session in a program, change in number of faculty required for a session, change in session time, session title and session commoncode. The DSS has been equipped to respond quickly and efficiently to all the above changes. A command file C.COM is called upon to make necessary changes automatically.

A Fortran program H99.FOR is called upon to find a good feasible solution in the revised scenario. For weeks that have gone by, the assignments are left untouched. This is done for purposes of record keeping and verification. For assignments that have become infeasible, suitable replacements are found. The objective as usual is to minimize perturbations. The user has the option of accepting the schedule generated by the program or using his experience to guide the program towards desired improvements.

4.2.5 Manual Fine Tuning

A Fortran program H50.FOR allows the user to carry out manual fine tuning of the schedule to account for subjective factors that are hard to quantify. This also enables the user to respond to a last minute request for a change. Relevant data about each instructor including total assignments made, total assignments made in the week, total assignments made in the day and availability are displayed to the user to enable him to make an informed decision.

4.3 LAGRANGIAN RELAXATION APPROACHES

In this section we discuss approaches for ISP that are based on the use of lagrangian relaxation techniques for solving integer programming problems. At first we present a discussion of the technique of lagrangian relaxation. Then we discuss upper bounds to ISP based on four different relaxations. Finally we present four heuristics that use the solution obtained from lagrangian relaxation and make necessary changes to develop a feasible solution.

4.3.1 A Discussion of Lagrangian Relaxation

In this section we present a discussion of the technique of lagrangian relaxation. This technique has become a popular one for solving integer programming problems. For a detailed discussion of the technique and its many applications we refer you to Fisher (1981), Frieze and Yadegar (1981), Jornsten and Nasberg (1986), Held, Wolfe and Crowder (1974), Muckstadt and Koenig (1977) and Geoffrion and McBride (1978).

The following discussion is based on Fisher (1981) with necessary modifications for a maximization problem.

Consider the following integer programming problem P:

$$Z = \text{Max } cx$$

subject to

$$b + Ax \geq 0$$

$$d + Bx \geq 0$$

$$x \geq 0, \text{ integer,}$$

where x is $n \times 1$, b is $m \times 1$, d is $p \times 1$, A is a matrix of coefficients of dimensions $m \times n$, B is a matrix of coefficients of dimensions $p \times n$, c is a row vector of costs of dimension n .

A lagrangian relaxation of P is the problem LR_u defined below.

Problem LR_u

$$Z_D(u) = \text{Max } cx + [u][b + Ax]$$

subject to

$$d + Bx \geq 0$$

$$x \geq 0, \text{ integer, } u \geq 0,$$

where u is a row vector of lagrangian multipliers of dimension m .

Observation

$$Z_D(u) \geq Z \text{ for all } u \geq 0.$$

Proof

Let x^* be an optimal solution to P. Thus, x^* is a feasible solution to LR_u , $u \geq 0$ and $[b + Ax^*] \geq 0$.

$$\text{Hence } Z_D(u) \geq cx^* + [u][b + Ax^*] \geq Z.$$

Thus for a given $u \geq 0$, $Z_D(u)$ is an upper bound on Z . We want to find the strongest upper bound. In other words we want to find $u \geq 0$ that would minimize $Z_D(u)$. This means that we want to find the optimal solution to the problem D as shown below.

$$Z_D = \min_u Z_D(u), \quad u \geq 0 \quad (D)$$

At first we discuss characteristics of D that make it possible for us to solve it efficiently.

Let X be the set of feasible solutions of LR_u . In other words, $X = \{x \mid dx + Bx \geq 0, x \geq 0, \text{integral}\}$. We assume that this set is finite and represent X as $X = \{x^k, k=1, \dots, K\}$. This permits us to rewrite problem D as a linear program $\bar{D}$ as shown below.

$$Z_D = \min w \quad (\bar{D})$$

subject to

$$w \geq cx^k + [u][b + Ax^k], \quad k = 1, \dots, K.$$

$$u \geq 0$$

For a given u , problem $\bar{D}$ says that w is at least as great as the largest right hand side in $\bar{D}$. This implies that $Z_D(u)$ is the upper envelope of a finite number of linear inequalities.

For the case of $m = 1$ and $K = 5$ the general form of $Z_D(u)$ is shown in figure 4.1. The function $Z_D(u)$ has the properties of continuity and convexity. However it is not differentiable at all points. Since the function is not continuously differentiable, we need to use techniques from non - differentiable optimization. The technique that has become popular in research is known as subgradient optimization and is presented in the next sub-section.

4.3.1.1 Subgradient Optimization

In this sub-section we present the steps in the sub-gradient algorithm. This algorithm is popular because it can be easily programmed and runs very fast. A complete description of the subgradient optimization algorithm and its convergence properties is presented in Held, Wolfe and Crowder (1974) and Goffin (1977).

In this algorithm a sequence of u vectors is generated that is expected to converge asymptotically to the best values of u , that is, the set of lagrange multipliers which yields the tightest bound. The algorithm finds a direction of search, computes a step size and then generates the next u vector. The steps in the algorithm are shown below.

Steps in Sub-Gradient Optimization

Notation:

k is the iteration number;

s_k is the step size in iteration k ;

u^k is the u vector in iteration k ;

t_k is a scalar in the interval $[0,2]$;

$\|f\|$ is the Euclidean Norm of f ;

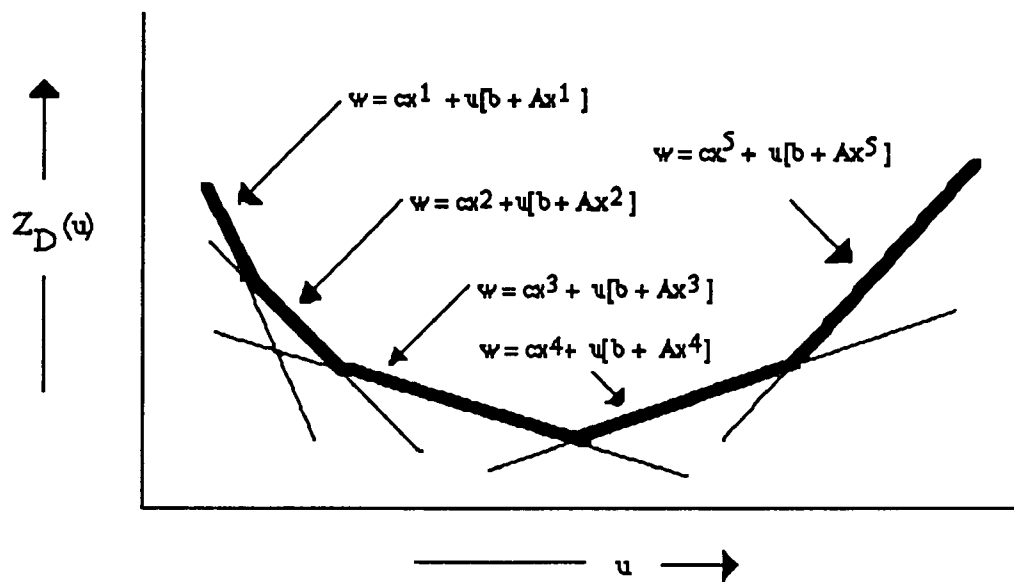


Figure 4.1 The form of $Z_D(u)$ for $m = 1$.

Step 1

Initialization:

$k = 0$;

$u^k = 0$; [a starting vector of 0 has worked well];

$t_k = 2$;

$ub = M$; [M is a very large number,e.g. sum of all costs];

lb = value of best solution found to P :[obtained by use of a heuristic];

x_k = optimal solution to problem LR_u in iteration k ;

Step 2

$k = k + 1$;

if $k > 300$, STOP; [terminate after 300 iterations];

Step 3

Solve LR_u and find $Z_D(u)$;

$ub = \min\{ ub, Z_D(u) \}$

Step 4

Compute a step size $s_k > 0$;

$$s_k = (t_k) * (Z_D(u) - lb) / [\| b + Ax_k \|^2]$$

Step 5

Find the new value of u ;

$$u^{k+1} = \max\{ 0, u^k + [s_k] * [b + Ax_k] \};$$

$[b_i + Ax_i]$ is the new direction of search for a particular row i .

Step 6

If ub has not decreased in several iterations,

$$t_{k+1} = 0.5 * t_k; \text{ else } t_{k+1} = t_k;$$

If $s_k = 0$, STOP; this u is optimal;

If $t_k = 0$, STOP;

4.3.2 Lagrangian Upper Bounds for ISP

In this section we discuss four upper bounds for the Instructor Scheduling Problem [ISP]. The basic approach is to relax a particular set of constraints in the problem in a lagrangian manner and then use subgradient optimization to find the best upper bound. In one of the relaxations we present a row generation approach that utilizes information from the violated constraints to generate equivalent constraints that are then relaxed in a lagrangian manner.

At first we present ISP in a form suitable for discussion of upper bounds.

Problem ISP

Maximize cx

subject to

TCS constraints [e.g. $XA1 + XA5 \leq 1$]

CC constraint [e.g. $XB1 - XB9 = 0$]

NETWORK constraints

$x \geq 0$, 0 or 1

The TCS constraints prevent assignment to two sessions that are in time conflict. The CC constraints stipulate that the pair of sessions must be taught by the same faculty. The NETWORK constraints are all the remaining constraints in the ISP. Recall that the remaining constraints are an example of SARS and hence can be formulated and solved as a network problem.

Upper Bound 1 [UB1]

This upper bound is obtained by relaxing TCS and CC constraints from the problem. The remaining problem is an example of SARS and is solved as a network by use of GNET.

Upper Bound 2 [UB2]

This upper bound is found by using the following steps.

- a] Ignore the CC constraints;
- b] Form a lagrangian relaxation of TCS constraints;
- c] Use subgradient optimization to find the best bound;

Upper Bound 3 [UB3] [A ROW GENERATION APPROACH]

This upper bound is found by using the following steps.

- a] Solve the network portion of the problem;
- b] Find violated CC constraints;
- c] Generate equivalent TCS type constraints [ETCS];
(Generate constraints equivalent to the violated rows);
- d] Relax TCS and ETCS constraints in a lagrangian manner;
- e] Use subgradient optimization to find the best bound;

Example of ETCS

Suppose that sessions 5 and 11 have the same commoncode. Suppose that instructors A and C are assigned to them when the network portion is solved in step [a]. In step [b] we find that this is a violation of the commoncode constraint: $XA005-XA011=0$. In step [c] we add an equivalent constraint of the form

$XA005 + XC011 \leq 1$. This constraint is of the type TCS. Hence we refer to it as an equivalent TCS or ETCS. The ETCS acts implicitly to prevent violation of a CC constraint. It does not act explicitly to guarantee that a particular CC constraint would be respected. Since TCS and ETCS are both of the same type, we treat them similarly in forming a lagrangian relaxation. Our earlier observation that an optimal solution of this relaxation is an upper bound on problem P remains valid as long as the lagrange multipliers for the relaxed TCS and ETCS constraints are non-negative.

Upper Bound 4 [UB4]

This upper bound is found by using the following steps:

- a) Same as steps [a] through [e] of procedure for finding UB3;
- b) Repeat steps [b] through [e] of procedure for finding UB3 five times.

The relaxed problems LR_u that we solve are all networks. These are solved by use of GNET (Bradley and Graves,1975). Since the lagrangian subproblem is a network, it has the integrality property. Hence, the value of the best upperbound would be equal to the optimal value of the linear programming relaxation of ISP (Fisher,1981).

4.3.3 Lagrangian Heuristics

In this section we discuss four lagrangian heuristics that may be used to generate "good" solutions to ISP. Lagrangian heuristics have become popular because of their ability to find "near optimal" solutions to many integer programming problems.

For references to work that report on the development and use of lagrangian heuristics see Fisher (1981). The basic idea behind development of these heuristics is

given below. A solution to the relaxed problem is feasible to network constraints. It often happens that such a solution is nearly feasible for ISP and can be made feasible with some necessary changes. Once a feasible solution is found it may be improved upon. Such an approach has been called a lagrangian heuristic.

Lagrangian Heuristic 1 [LH1]

The steps in the heuristic are summarized below:

- a) Execute procedure for finding UB1;
- b) Use JTR to find a feasible solution and improve upon it.

Lagrangian Heuristic 2 [LH2]

The steps in the heuristic are summarized below:

- a) Execute procedure for finding UB2;
- b) Use JTR to find a feasible solution and improve upon it.

Lagrangian Heuristic 3 [LH3]

The steps in the heuristic are summarized below:

- a) Execute procedure for finding UB3;
- b) Use JTR to find a feasible solution and improve upon it.

Lagrangian Heuristic 4 [LH4]

The steps in the heuristic are summarized below:

- a) Execute procedure for finding UB4;
- b) Use JTR to find a feasible solution and improve upon it.

The computational performance of these heuristics are presented in Chapter V.

The above heuristics use a sub-routine JTR. Given an infeasible solution, sub-routine JTR makes necessary changes to find a feasible solution and then improves upon it. In particular, sub-routine JTR identifies every session that is infeasible because of either CC constraints or TCS constraints and releases the instructor assigned to these sessions. For each such session all the feasible instructors are identified and the best in terms of objective function value is assigned to the session. When a feasible assignment is found, two faculty interchanges and three faculty interchanges are carried out to improve the solution if such an interchange results in an increase in the objective function value.

CHAPTER V

COMPUTATIONAL EXPERIMENTS

This chapter summarizes the design and results of the computational experiments. Solutions found by the 12 knowledge based heuristics are compared to the optimal solution of 81 test problems and 2 real problems. The heuristic that uses a schedule generation strategy based on selecting faculty with higher benefit and the combination schedule improvement strategy is found to be the best. Sensitivity of the heuristic solutions to various benefit levels is investigated. It is found that average percentage deviation from optimal [APD] remains below 1.6% for the best three heuristics when different values are used for the three different benefit levels. The APD of the best heuristic drops to 0.57% when benefits are drawn at random.

Computational experiments on lagrangian approaches for finding upper bounds and heuristic solutions are reported. In problems with random benefits, the best upper bounding procedure finds a bound that is 100.12% of the optimal objective value on average and equal to the optimal objective value in 10 out of 27 problems. The best lagrangian heuristic finds solutions with an APD of 0.28% from optimal and equal to the optimal solution in 13 out of 27 problems.

Managerial implications of computational results are discussed. The problem appears to become more difficult when benefit values have large differences and the percentage of commoncode sessions is large. It is suggested that the preliminary

schedule should be generated by any reasonable method and the greater stress should be on designing effective strategies to improve upon the schedule.

A hierarchical strategy for responding to the planning needs of the scheduler is proposed. This strategy proposes use of the optimal solution for generation of the first plan, monthly updating by use of the lagrangian heuristics and more frequent updating by use of the knowledge based heuristics.

5.1 DESIGN OF EXPERIMENTS FOR TEST PROBLEMS

In this section we describe the test problems that were used to investigate performance of the knowledge based heuristics. The computational study consists of finding the exact and heuristic solutions of 81 test problems. Each of the 81 test problems are solved by the 12 heuristics and also by LINDO to get their optimal values.

The test problems differ in four parameters. Each of these four parameters could assume one of three possible values. This yielded a total of $3^4 = 81$ possible combination of parameter values. A test problem was generated for each such combination.

The four problem parameters are (1) Number of Sessions, (2) Number of Faculty, (3) Percentage of Common-Code Sessions and (4) Level of Faculty Inclination. The Level of Faculty Inclination [LFI] indicates the proportion of faculty that shows strong inclination for particular sessions. For example, a low LFI indicates

that a small proportion of faculty shows strong preferences for teaching sessions. LFI may be low, medium or high. The percentage of faculty with different benefit points for different levels of LFI is shown in Table 5.1. The three possible values for each parameter are shown in Table 5.2.

The lower limit and upper limit on total assignments to a faculty are randomly drawn from uniform distributions whose ranges are tabulated in Table 5.3. One faculty is required for each session.

Twelve heuristics are tested in this computational experiment. Each of the heuristics uses a different combination of schedule generation and schedule improvement strategies. The different combinations of strategies used by the heuristics are shown in Table 5.4.

5.2 RESULTS OF TEST PROBLEMS

The twelve heuristics were incorporated in a Fortran program and tested on the University of Tennessee's VAXCluster. The exact solution to the test problems were found by using LINDO. We compare the performance of the 12 heuristics on the basis of the average of the percentage deviation from the optimal solution and the maximum percentage deviation from the optimal solution. The performance of the best heuristic with respect to variation over the parameter values is discussed.

Table 5.1 Percentage of faculty with different benefit points for different levels of faculty inclination

LFI ^a	<u>% of faculty with different benefit points</u>		
	<u>B E N E F I T P O I N T S</u>		
	10	20	40
Low	70%	20%	10%
Medium	40%	30%	30%
High	10%	20%	70%

^aLevel of Faculty Inclination

Table 5.2 Values of Different Parameters

Parameter	Level 1	Level 2	Level 3
Number of Sessions	50	100	150
Number of Faculty	6	8	10
% Common-code Sessions	20%	30%	40%
LFI ^a	low	medium	high

^aLevel of Faculty Inclination

Table 5.3 Range of Uniform Distributions from which
Total Limits of a Faculty are Drawn.

Number of Sessions	Range for			Range for		
	<u>Lower Limits</u>			<u>Upper Limits</u>		
	<u>Number of Faculty</u>			<u>Number of Faculty</u>		
	6	8	10	6	8	10
50	1-5	1-4	1-3	25-35	15-25	10-20
100	2-6	3-7	1-4	40-50	28-38	25-35
150	2-8	3-7	1-5	60-70	50-60	25-35

Table 5.4 Strategies used in different Heuristics for Schedule Generation and Schedule Improvement

HNa	Generation Strategy	Improvement Strategy
1	Select lowest proportion assigned	2 faculty replacement
2	Select lowest proportion assigned	2 faculty interchange
3	Select lowest proportion assigned	3 faculty interchange
4	Select lowest proportion assigned	Combination
5	Select highest benefit	2 faculty replacement
6	Select highest benefit	2 faculty interchange
7	Select highest benefit	3 faculty interchange
8	Select highest benefit	Combination
9	Select highest composite score	2 faculty replacement
10	Select highest composite score	2 faculty interchange
11	Select highest composite score	3 faculty interchange
12	Select highest composite score	Combination

^aHeuristic Number.

5.2.1 Comparison of 12 Heuristics

All the 81 test problems were solved optimally by use of LINDO. The problems were solved by each of the 12 heuristics and percentage deviation (PD) of the heuristic solution from the optimal solution was found. The minimum PD, maximum PD and average PD for the 12 heuristics are shown in Table 5.5.

It is found that heuristic number 8 has performed the best in terms of average percentage deviation (1.14%). Heuristic number 4 and 12 are second and third best in terms of average percentage deviation (1.17% and 1.22% respectively).

5.2.2 Variation of Performance over Parameter Values

For each of the 4 problem parameters and each of the 3 levels of a parameter, the average percentage deviation (APD) between the optimal value and the heuristic value has been found for each of the 12 heuristics. These are shown in Tables 5.6 through 5.9.

The variation of performance over parameter values is discussed with respect to the performance of the best heuristic, that is, heuristic number 8. Changes in the level of faculty inclination (LFI) had the most significant effect on APD. There was a decreasing trend as LFI changed from low to medium and from medium to high. Changes in number of faculty had the second most significant effect on APD. Changes in number of sessions had the third most significant effect on APD. Variation in APD due to changes in percentage of commoncode sessions were found to be small.

Table 5.5. Comparison of 12 Heuristics on the basis of
Percentage Deviation from the Optimal Solution.

HN ^a	Minimum	Maximum	Average
1	0.00	6.02	1.59
2	0.00	14.81	5.36
3	0.00	16.89	3.86
4	0.00	4.76	1.17
5	0.00	7.33	1.99
6	0.55	8.99	3.83
7	0.00	7.33	2.58
8	0.00	5.18	1.14
9	0.00	7.58	2.23
10	0.76	23.61	6.81
11	0.00	22.92	5.51
12	0.00	5.64	1.22

^aHeuristic Number.

Table 5.6 Variation in Average Percentage Deviation
as Number of Sessions Change

HN ^a	NUMBER OF SESSIONS		
	50	100	150
1	2.00	1.19	1.56
2	6.84	4.90	4.33
3	6.42	3.18	1.99
4	1.37	0.97	1.16
5	3.13	1.60	1.24
6	3.77	3.87	3.87
7	2.96	2.59	2.19
8	1.36	1.04	1.02
9	2.99	2.08	1.62
10	8.33	6.33	5.76
11	7.72	5.12	3.68
12	1.46	1.32	0.90

^aHeuristic Number

Table 5.7 Variation in Average Percentage Deviation as Percentage of Commoncode Sessions Change

HN ^a	PERCENTAGE OF COMMONCODE SESSIONS		
	20%	30%	40%
1	1.33	1.13	1.79
2	4.63	4.07	5.81
3	4.67	2.79	3.46
4	1.00	0.69	1.45
5	1.67	1.78	2.10
6	2.79	2.68	4.69
7	2.17	1.93	2.90
8	1.09	0.88	1.10
9	2.05	1.72	2.38
10	5.90	4.84	7.82
11	4.82	4.32	6.13
12	1.22	0.88	1.27

^aHeuristic Number

Table 5.8 Variation in Average Percentage Deviation
as Number of Faculty Changes

HN ^a	NUMBER OF FACULTY		
	6	8	10
1	2.34	0.82	1.60
2	5.30	4.75	6.02
3	4.31	3.80	3.47
4	1.61	0.62	1.27
5	1.52	2.07	2.37
6	3.57	3.73	4.20
7	2.91	1.86	2.98
8	1.16	0.89	1.37
9	2.85	1.97	1.87
10	6.43	7.77	6.22
11	5.81	6.24	4.47
12	1.48	1.15	1.04

^aHeuristic Number

Table 5.9 Variation in Average Percentage Deviation
as Level of Faculty Inclination Changes

HN ^a	LEVEL OF FACULTY INCLINATION		
	low	medium	high
1	2.80	1.61	0.34
2	8.30	5.81	1.96
3	6.23	3.95	1.41
4	1.94	1.27	0.29
5	2.27	2.26	1.44
6	4.45	4.84	2.22
7	3.07	3.62	1.06
8	1.82	1.45	0.15
9	3.06	2.25	1.38
10	11.14	6.78	2.50
11	8.37	6.37	1.78
12	1.78	1.42	0.46

^aHeuristic Number

The APD was highest for low LFI (1.82%). It was lowest for high LFI (0.15%). It rose to be the second highest for medium LFI (1.45%). This is shown in Figure 5.1.

The APD was highest when number of faculty was 10 (1.37%). It was lowest when number of faculty was 8 (0.89%). It was second highest when the number of faculty was 6 (1.16%). This is shown in Figure 5.2.

The effects on APD due to Changes in Number of Sessions and Percentage of Commoncode Sessions appears to be small (see Figures 5.3 and 5.4).

5.3 SENSITIVITY ANALYSIS

In this section we discuss the sensitivity of heuristic performance to variations in the value of benefit parameters. It is found that the heuristics perform consistently well. The best 3 heuristics had average percentage deviations from optimal of 1.6% or less in all the scenarios investigated.

5.3.1 Relative Ratio of Benefits [RRB]

At present, the benefit values are at 3 levels: (a) Level 1 has a benefit of 10, (b) Level 2 has a benefit of 20, and (c) Level 3 has a benefit of 40. In this case the Level 2 benefit is twice that of Level 1 benefit and Level 3 benefit is twice that of Level 2. In other words the ratio between benefits of Level 1, Level 2 and Level 3 is 1:2:4.

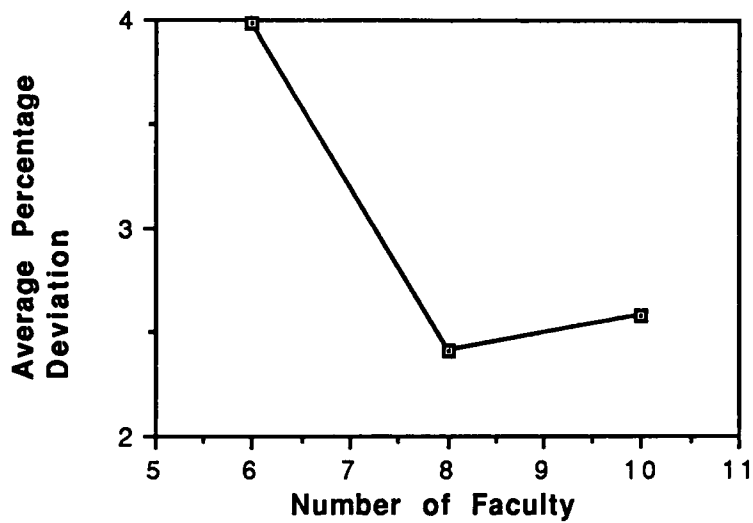


Figure 5.1 Effect on Average Percentage Deviation due to Changes in Level of Faculty Inclination

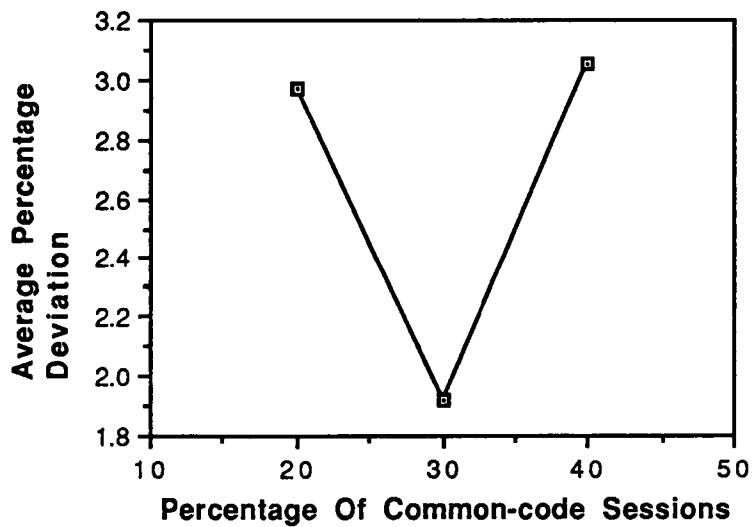


Figure 5.2 Effect on Average Percentage Deviation due to Changes in the Number of Faculty.

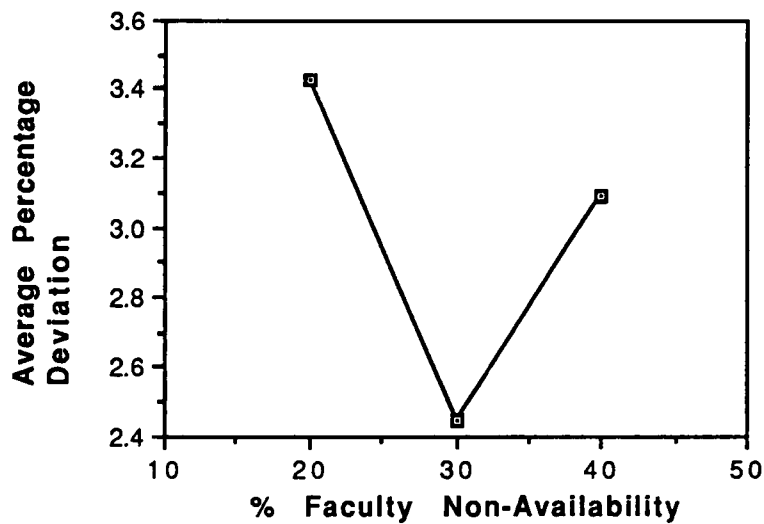


Figure 5.3 Effect on Average Percentage Deviation due to Changes in Number of Sessions.

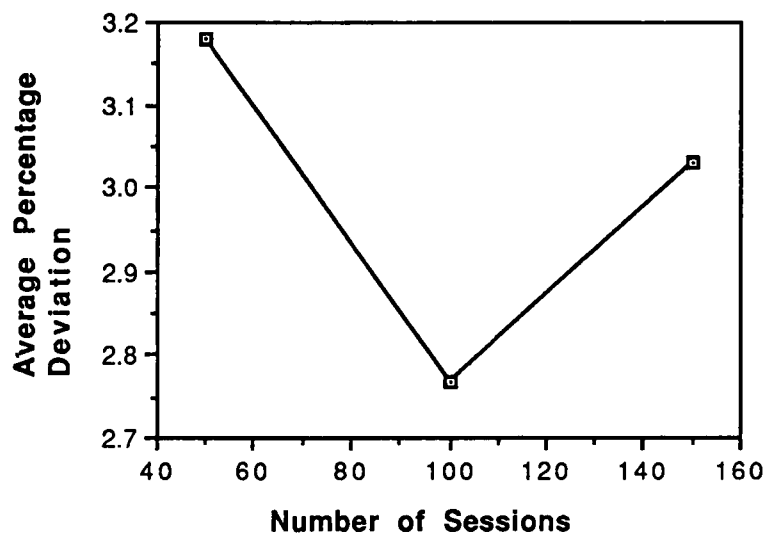


Figure 5.4 Effect on Average Percentage Deviation due to Changes in % of Commoncode Sessions

5.3.2. Three Cases That Arise

In order to evaluate heuristic performance when different benefit values are used we need to look at 3 cases.

Case 1: RRB Remains Same:

This may happen if all benefit points are scaled (multiplied or divided) by a common positive number. Since every benefit value is scaled by the same factor, the solution remains unaffected. The final objective value obtained from LINDO or any heuristic is scaled by the same common positive number. Thus the percentage deviation of heuristic solution from optimal remains the same.

Case 2 RRB is changed marginally

This may happen when all benefits are changed by a common number d which may be positive or negative but is small in relation to the smallest original benefit value of 10. Hence change in RRB is considered to be marginal. For example, we consider the 2 cases A1 and A2 where d takes on the values +2 and -5 respectively. The benefit values in these 2 cases is shown below.

BENEFITS

<u>CASE</u>	<u>LEVEL1</u>	<u>LEVEL2</u>	<u>LEVEL3</u>
A1	12	22	42
A2	5	15	35

The performance of the 12 heuristics in the above 2 cases is shown in Table 5.10 and Table 5.11. In the following discussion the performance of heuristics for benefit points of 10, 20 and 40 will be referred to as the BASE. Average Percentage Deviation will be referred to as APD.

In case A1, heuristics 8, 4 and 12 turn out to be the best performers as in the BASE. The APD for heuristic 8 improves to 1.07% as compared to 1.14% in the BASE. The APDs for heuristics 4 and 12 also show improvements of 0.07% over the BASE. These are small changes in APD.

In case A2, heuristics 12, 8 and 4 turn out to be the best performers. Although heuristic 8 is outperformed by heuristic 12, the difference in average percentage deviation is only 0.03%. The APD for heuristic 12 has worsened by 0.10% to 1.32% as compared to 1.22% in the BASE. The APDs for heuristic 4 and 8 have worsened by 0.21% each. These are small changes in APD. Another interesting point is that heuristics that use a combination strategy for improvement are still the best performers.

Table 5.10 Performance of Heuristics when benefit values
of 12, 22 and 42 are used

HN ^a	<u>Percentage Deviation from Optimal Solution</u>		
	Minimum	Maximum	Average
1	0.00	5.60	1.49
2	0.00	14.08	5.05
3	0.00	15.84	3.64
4	0.00	4.43	1.10
5	0.00	6.97	1.88
6	0.52	8.55	3.62
7	0.00	6.97	2.44
8	0.00	4.93	1.07
9	0.00	7.05	2.10
10	0.72	22.11	6.41
11	0.00	21.46	5.19
12	0.00	5.25	1.15

^aHeuristic Number

Table 5.11 Performance of Heuristics when benefit values
of 5, 15 and 35 are used

HN ^a	<u>Percentage Deviation from Optimal Solution</u>		
	Minimum	Maximum	Average
1	0.00	7.40	1.88
2	0.00	17.02	6.33
3	0.00	20.24	4.57
4	0.00	5.86	1.38
5	0.00	8.41	2.33
6	0.63	10.33	4.50
7	0.00	8.41	3.03
8	0.00	5.93	1.35
9	0.00	6.09	2.06
10	0.86	16.05	6.72
11	0.00	15.61	4.88
12	0.00	5.95	1.32

^aHeuristic Number

These results suggest that the heuristics are not very sensitive to marginal changes in the relative ratio between benefits.

Case 3 RRB is changed more than marginally

We consider cases with relative ratio of benefits of 1:3:9 and 10:11:12. In particular we consider the two cases A3 and A4 with benefits of different levels as shown below.

<u>B E N E F I T S</u>			
<u>CASE</u>	<u>LEVEL1</u>	<u>LEVEL2</u>	<u>LEVEL3</u>
A3	10	30	90
A4	10	11	12

The performance of the 12 heuristics in the above 2 cases is shown in Table 5.12 and Table 5.13. In case A3, heuristics 12,4 and 8 turn out to be the best performers. Although heuristic 8 is outperformed by heuristics 12 and 4, the difference in average percentage deviation is only 0.04% and 0.03% respectively. The APD for heuristic 12 has worsened by 0.30% to 1.52% as compared to 1.22% in the BASE. The APDs for heuristic 4 and 8 have worsened by 0.36% and 0.42% respectively. However, the average percentage deviations of the three best heuristics have remained within reasonable limits.

Table 5.12 Performance of Heuristics when benefit values
of 10, 30 and 90 are used

HN ^a	<u>Percentage Deviation from Optimal Solution</u>		
	Minimum	Maximum	Average
1	0.00	7.97	2.11
2	0.00	18.18	6.96
3	0.00	21.85	5.05
4	0.00	6.17	1.53
5	0.00	8.96	2.65
6	1.00	11.48	4.90
7	0.00	9.43	3.40
8	0.00	6.51	1.56
9	0.00	7.00	2.36
10	0.90	17.52	7.22
11	0.00	16.87	5.54
12	0.00	6.02	1.52

^aHeuristic Number

Table 5.13 Performance of Heuristics when benefit values
of 10, 11 and 12 are used

HN ^a	<u>Percentage Deviation from Optimal Solution</u>		
	Minimum	Maximum	Average
1	0.00	1.05	0.29
2	0.00	2.86	0.97
3	0.00	2.98	0.68
4	0.00	1.05	0.23
5	0.00	1.51	0.35
6	0.00	1.68	0.68
7	0.00	1.35	0.49
8	0.00	1.01	0.20
9	0.00	1.16	0.38
10	0.17	3.70	1.21
11	0.00	3.55	0.93
12	0.00	0.80	0.21

^aHeuristic Number

In case A4, heuristics 8, 12 and 4 turn out to be the best performers. The APD of heuristic 8 improved from 1.14% to 0.20%. The APDs of heuristic 12 and 4 improved from 1.22% and 1.17% to 0.21% and 0.23% respectively. The changes were all in directions of improvement.

These results seem to suggest that more than marginal changes in the relative ratio of benefits may bring about changes in the APD. However, the performance of the best three heuristics appear to be consistent and the average percentage deviation remains small.

5.4 HEURISTIC PERFORMANCE WITH RANDOM BENEFITS

In this section we discuss the performance of the heuristics when the benefits are sampled randomly. Heuristic number 8 turns in the best performance with an average percentage deviation [APD] of 0.57% from optimal. The performance in problems with random benefits appear to be better than the performance when there were three levels for the benefit.

The test problems vary according to three parameters. These parameters are (a) Number of Faculty, (b) Number of sessions, and (c) percentage of commoncode sessions. Each of the parameters can take on three values as shown in Table 5.2. For each combination of parameter values a problem is generated. The benefit values are drawn randomly from the uniform distribution in the range 1 to 99.

The performance of the heuristics is shown in Table 5.14. From an APD of 1.14% in the BASE, heuristic number 8 improved its performance to an APD of 0.57%.

Heuristic number 12 improved its performance from an APD of 1.22% to 1.02%. Heuristic number 4 had an APD of 1.58% which was worse than its BASE APD of 1.17%. Heuristic number 5 turned in the second best APD of 0.78%. Overall, it would appear that the heuristics have turned in a better performance with random benefits.

5.5 PERFORMANCE OF HEURISTICS ON REAL PROBLEMS

The heuristics performed reasonably well on the two real problems. Two real problems were solved optimally and with the use of the 12 heuristics. Performance of the heuristics in terms of objective function value, percentage difference and CPU time are discussed.

Data is available for the planning periods of Fall (1988) and Spring (1989). The first problem is referred to as FA88 and the second problem is referred to as SP89. The problem FA88 has 547 sessions and 14 faculty. It was solved by LINDO in 48 minutes 52.38 seconds of CPU time. The corresponding wall clock time taken was 1 hour 19 minutes and 50.78 seconds. The wall clock time is important because it is this time that the user experiences. The problem was also solved by the 12 heuristics.

Table 5.14 Performance of Heuristics when benefit values
are sampled randomly

HN ^a	<u>Percentage Deviation from Optimal Solution</u>		
	Minimum	Maximum	Average
1	0.07	4.26	1.39
2	2.17	13.73	5.71
3	1.38	11.66	4.83
4	0.07	4.53	1.58
5	0.02	1.65	0.78
6	1.03	7.46	3.51
7	0.49	5.85	2.47
8	0.02	1.87	0.57
9	0.11	2.94	1.07
10	3.33	12.02	7.27
11	0.74	17.06	6.70
12	0.02	2.85	1.02

^aHeuristic Number

The maximum CPU time taken by any heuristic was 23.35 seconds. For this problem heuristic number 8 found the best solution which was 4.26% less than the optimal value of 12920 found by LINDO. Heuristic number 8 used 23.35 CPU seconds to find this solution. The problem SP89 has 402 sessions and 14 faculty. The best integer solution found by LINDO had a value of 14740. LINDO evaluated the current upper bound to be 15000. However, it could neither find a better solution nor prove optimality although the problem was run for over 16 hours of CPU time (24 hours and 54 minutes of wall clock time). In practice, the user would use an integer programming tolerance factor IPTOL available on LINDO. For example, if the user specifies an IPTOL of 0.03, LINDO would terminate the procedure if it finds an integer solution within 3% (or IPTOL %) of the current upper bound. The problem SP89 was submitted to LINDO with an IPTOL of 0.025. LINDO found an integer solution of 14680 and terminated the procedure in 24 minutes of CPU time. The maximum CPU time taken by any of the 12 heuristics on this problem was 13.64 seconds. Heuristic number 4 used 7.99 CPU seconds to find the best solution of 14380. This was 4.13% away from the current upper bound of 15000. The solution values found by each of the 12 heuristics on the 2 real problems along with percentage deviation from the optimal/current upper bound is shown in Table 5.15. Heuristic number 8 is seen to have performed the best with an average percentage deviation of 5.93%. Heuristic number 7 is a close second with an average percentage deviation of 5.95%.

Table 5.15 Performance of the Heuristics on the Real Problems

HN ^a	FALL 1988		SPRING 1989		APD ^e
	Soln. Val. ^b	% Devn. ^c	Soln. Val.	% Devn. ^d	
1	10480	18.89	13340	11.07	14.98
2	11630	9.98	13470	10.20	10.09
3	11710	9.37	13940	7.07	8.22
4	11720	9.29	14380	4.13	6.71
5	10620	17.80	13470	10.20	14.00
6	10660	17.49	13650	9.00	13.25
7	12330	4.57	13900	7.33	5.95
8	12370	4.26	13860	7.60	5.93
9	10590	18.03	13380	10.80	14.42
10	10610	17.88	13390	10.73	14.31
11	11460	11.30	14050	6.33	8.82
12	11910	7.82	13840	7.73	7.78

^aHeuristic Number.

^bSolution Value found by heuristic.

^cThis shows percentage deviation from optimal value of 12920.

^dThis shows percentage deviation from current upper bound of 15000.

^eAverage Percentage Deviation.

From a managerial perspective, the schedules generated by the heuristics had similarities and differences with those generated by the optimal approach. The schedules were similar in the extent of fine tuning that was needed to make the schedules viable. About 10% to 20% of the assignments needed to be changed by the scheduler in order to make them viable. Further, neither approach did a good job of accounting for incompatibility of faculty when assigning two or more of them to the same session. The schedules were different in that the actual assignments made to the sessions were rarely the same.

5.6 LAGRANGIAN APPROACHES

Computational experiments on the lagrangian approaches consisted of testing the upper bound procedures and lagrangian heuristics on 27 random problems and two real problems. In the random problems the best upper bound procedure UB4 finds bounds that are 100.12% of the optimal on an average. Procedure UB4 finds a bound equal to the optimal in 10 out of 27 problems. In the random problems the best heuristic LH4 finds solutions that have an APD of 0.28% from the optimal solution. Heuristic LH4 finds the optimal solution in 13 out of 27 problems.

Random Problems

The 27 random problems and random benefits are generated in the same way as discussed in section 5.4.

The performance of the upper bound procedures and lagrangian heuristics are summarized in Table 5.16 and Table 5.17 respectively.

Lagrangian upper bound procedures UB2, UB3 and UB4 find bounds that are 100.14%, 100.14% and 100.12% of the optimal solution on an average. Procedures UB2, UB3 and UB4 find bounds equal to the optimal in 7 out of 27, 8 out of 27 and 10 out of 27 problems, respectively.

Lagrangian heuristics LH1, LH2, LH3 and LH4 find solutions with APDs of 0.87%, 0.52%, 0.45% and 0.28% respectively. Heuristics LH3 and LH4 both find the optimal solution in 13 out of 27 problems.

Real Problems

The lagrangian procedures were tested on the two real problems discussed in section 5.5.

The performance of the upper bound procedures and lagrangian heuristics are summarized in Table 5.18 and Table 5.19 respectively.

For both the problems procedure UB4 finds the best bound. For the data of Fall 1988 the best bound found is 100.13% of optimal. For Spring,1989 the best bound found is 102.24% of the best integer solution of 14740 found by LINDO.

For the data of Fall 1988, heuristic LH4 finds the best solution of 12490 that is 96.67% of the optimal. For Spring 1989, heuristic LH2 finds the best solution of 14480 that is 98.23% of the best integer solution.

Table 5.16 Performance of lagrangian upper bounding procedures on random problems

Procedure	Average % of Optimal
UB1	100.48
UB2	100.14
UB3	100.14
UB4	100.12

Table 5.17 Performance of lagrangian heuristics on random problems

Heuristic	Percent Deviation from Optimal		
	Minimum	Maximum	Average
LH1	0.00	6.68	0.87
LH2	0.00	5.10	0.52
LH3	0.00	5.10	0.45
LH4	0.00	2.44	0.28

Table 5.18 Performance of lagrangian upper bound procedures on real problems

Procedure	<u>FALL 1988</u>		<u>SPRING 1989</u>	
	Value [cpu time]	POO ^a	Value [cpu time]	POO ^b
UB1	12970 [25sec]	100.38%	15160 [21 sec]	102.84%
UB2	12970 [6min 6sec]	100.38%	15101 [8 min 30 sec]	102.44%
UB3	12963.90 [8min 32.1sec]	100.34% [9 min 55 sec]	15090	102.37%
UB4	12937.14 [20 min 13.5sec]	100.13%	15071.37 [22min 6.7 sec]	102.24%

^aPercent of optimal value of 12920

^bPercent of best integer solution of 14740 found by
LINDO

Table 5.19 Performance of lagrangian heuristics
on real problems

Procedure	<u>FALL 1988</u>		<u>SPRING 1989</u>	
	Value [cpu time]	POO ^a	Value [cpu time]	POO ^b
LH1	12390 [26sec]	95.89%	14290 [18 sec]	96.94%
LH2	12390 [6min 19sec]	95.89%	14480 [8 min 20 sec]	98.23%
LH3	12440 [8min 27sec]	96.28%	14410 [10 min 10 sec]	97.76%
LH4	12490 [59 min 17sec]	96.67%	14450 [28 min 17 sec]	98.03%

^aPercent of optimal value of 12920

^bPercent of best integer solution of 14740 found by
LINDO

5.7 DISCUSSION OF COMPUTATIONAL RESULTS

In this section we discuss managerial implications of the computational results. The problem appears to become more difficult when (a) benefit values have large differences, (b) benefit values are not distinct, (c) few faculty are strongly inclined, (d) few sessions need to be staffed and (e) the percentage of commoncode sessions is large.

The Average Percentage Deviation (APD) were found to be the smallest when benefit values of 10, 11 and 12 were used. This probably happens because the penalty in objective function for making a wrong choice is small when the benefit levels are close to each other. This suggests the suitability of ranking the instructors for each session.

If we exclude the case when benefit values of 10,11 and 12 were used, the APD's of the best knowledge based heuristics (KBH), that is heuristic 8 and 12, are lower with random benefits than with three benefit levels. This suggests that solutions with lower APD are likely to be obtained when benefit levels are distinct. In other words, since the APD is generally higher when three levels of benefit values are used, these problems can be regarded as more difficult.

The APD for ten out of twelve KBH heuristics decreases as the Level of Faculty Inclination [LFI] changes from low to medium and then from medium to high (Table5.9). For the scheduler this implies that it is easier to develop high quality schedules if more faculty are strongly inclined. Heuristics 8, 4 and 12 are the best

performers when LFI is low or medium. However, heuristics 8, 4 and 1 are the best three in terms of APD when LFI is high.

The APD for nine out of twelve KBH heuristics decreases as the number of sessions increases from 50 to 100 and then from 100 to 150 (Table5.6). This suggests that an increase in number of sessions generally makes it easier to find high quality solutions. Heuristics 8, 4 and 12 are the best three when there are 50 sessions or 150 sessions. However, when there are 100 sessions, heuristics 8, 4 and 1 are the best three performers.

The APD for eleven out of twelve KBH heuristics drops and then increases as percentage of commoncode sessions changes from 20% to 30% and then from 30% to 40% (Table5.7). The increase in APD as percentage of commoncode sessions changed from 30% to 40% is much higher than the drop in APD as percentage of commoncode sessions changed from 20% to 30% in eleven out of twelve heuristics. It would appear that some kind of minima was observed near 30% commoncode sessions. This suggests that as percentage of commoncode sessions increases beyond 30% it is more difficult to get good quality solutions. The best three performers at all levels of percentage of commoncode sessions are heuristics 4, 8 and 12.

The behavior of the various heuristics as the number of faculty changes is not very clear. As the number of faculty increases from 6 to 8 and then from 8 to 10, (a) five heuristics show a decline in APD followed by an increase in APD, (b) three heuristics show a continuous decline in APD, (c) two heuristics show a continuous increase in APD and (d) two heuristics show an increase in APD followed by a decrease in APD (Table5.8). Heuristics 8, 12 and 5 are the best three when the number of faculty is equal to six. When the number of faculty increases to eight, heuristics 4, 1

and 8 are the best three. However, when the number of faculty increases to ten, heuristics 12, 4 and 8 turn out to be the best three.

In the overall analysis of heuristic performance as parameter values were varied, heuristics 8, 12 and 4 were found to have performed well consistently.

If we compare the performance of the twelve knowledge based heuristics, it would appear that the best performers have all used a combination improvement strategy. This suggests (a) that the preliminary schedule should be generated by any reasonable method that does not require too much computational effort, and (b) the emphasis should be on designing effective strategies to improve upon the preliminary schedule.

The best heuristics i.e. heuristics 8, 12 and 4 have all used a combination improvement strategy. This strategy uses two-faculty-replacements, two-faculty-interchanges and three-faculty-interchanges in sequence to improve the schedule. We would like to ask why these improvement strategies produce near optimal solutions. We believe that the replacement and interchange strategies attain local optima which cumulates subsequently into a solution that is close to the global optima because the constraint space is not very tight. This, in turn, implies that many alternate optimal solutions are very likely to exist. From the point of view of a decision maker, this is generally advantageous because it provides the flexibility to select a viable schedule from many equally effective ones.

Among the lagrangian heuristics, LH4 turns in the best performance both for random and real problems. However, LH4 takes an average of 40 minutes on the two real problems to find solutions with an APD of 2.6%. In contrast, heuristics LH2 and

LH3 take an average of 7 minutes and 9 minutes respectively to find solutions with an APD of 2.9% each. The gain of 0.3% in APD takes place at the expense of an additional 30 minutes of computer time. This suggests that the extra effort applied in LH4 for carrying out 5 iterations of procedure UB3 is probably not justifiable in a practical situation. Strictly from a cost-benefit point of view, a manager would find it more desirable to use heuristic LH2 or LH3.

5.8 A HIERARCHICAL STRATEGY FOR MEETING PLANNING

NEEDS OF THE SCHEDULER

In this section we propose a hierarchical strategy for responding to the planning needs of the scheduler. The scheduling environment is dynamic and requires the scheduler to respond to changes in the environment. At first we discuss the nature of the scheduling task.

A schedule is prepared in advance for a planning period of four months. At the end of each month, changes to the next months' schedule are circulated. When a particular month is in progress, there are occasions that demand changes in the schedule for the next week or the next day or some other day in the near future. The nature of the task is thus hierarchical and we propose an appropriate strategy to meet the needs of the scheduler.

A Hierarchical Strategy

The strategy that we propose consists of use of an optimal approach for generating the first schedule, use of lagrangian heuristics for monthly updating of

schedule and use of knowledge based heuristics for adjustments to the current month's schedule. Details of the strategy are presented below.

About four weeks before the beginning of a planning period the scheduler will develop an optimal schedule. This would yield the best schedule under the specified constraints. This optimal schedule will act as a "reference" schedule and changes, if necessary, may be made as long as perturbations to the "reference" schedule are minimized. The optimal model would probably require over 30 minutes of CPU time and should be solved as a batch job.

Two weeks before start of each month, lagrangian relaxation heuristics (LH2 or LH3) may be used to find updated schedules for the forthcoming month. These heuristics are expected to require one to three minutes of CPU time when working on a monthly schedule.

As and when changes are required in the current month's schedule, the knowledge based heuristics may be used to make suitable changes in the schedule. These heuristics would typically require less than 5 seconds of CPU time when revising the current month's schedule. Thus it would be possible to carry out several runs to investigate alternative scenarios and develop contingency plans for requesting different instructors.

CHAPTER VI

IMPLEMENTATION ISSUES

In this chapter we discuss implementation issues. Issues relating to reduction in size of the generated mathematical program, use of LINDO in solving the problem and verification of the mathematical program are discussed. A sample mathematical program for two instructors and four sessions is shown. Input and output data processing is discussed.

6.1 REDUCTION IN PROBLEM SIZE

In this section strategies implemented for reduction in problem size are discussed. In particular, variables that must take on zero values in the final solution are not created and wholly redundant constraints are eliminated.

6.1.1 Reduction in Number of Variables

The number of variables is reduced by not creating them under the following circumstances:

- a) If an instructor is not available to teach all the sessions in a set of commoncode sessions, the corresponding variables are not created.
- b) If instructor I has been committed to session J, variables of other instructors for session J are not created.
- c) If instructor I has been committed to a commoncode session J, instructor I is committed to the other sessions with the same commoncode and variables of other instructors for this set of commoncode sessions are not created.

d) If instructor I has been committed to session J, then variables for this instructor for sessions in time conflict with J are not created. If J is a commoncode session then variables for sessions in time conflict with this set of commoncode sessions are not created.

6.1.2 Reduction in Number of Constraints

The number of constraints is reduced by eliminating them under the following circumstances:

- a) If they are duplicate constraints they are eliminated.
- b) If an instructor is not available for one of a pair of sessions in time conflict then the corresponding constraint to prevent time conflicts is not written.
- c) A commoncode constraint that is implied by presence of other commoncode constraints is not written.

example:

Suppose sessions 1, 2 and 3 have the same commoncode. Suppose that instructors 1 and 2 are available to teach all the three sessions. Then the constraints added take the form:

$$XA001 - XA002 = 0$$

$$XA001 - XA003 = 0$$

$$XA002 - XA003 = 0 \quad \text{.....(A)}$$

Constraint A is implied by the previous two constraints and is not written.

d) If an instructor is not available for more than three sessions on any one day the corresponding constraint to prevent assignment of more than three sessions on any day is not written.

6.1.3 Results of Use of Size Reduction Strategies

Reduction in problem size that resulted from use of the above strategies in generation of the mathematical program for the data of Fall 1988 is shown below:

a) About 1800 variables out of a total of 7658 zero-one integer variables were not created.

b) About 23400 constraints out of a total of 28300 constraints were eliminated.

6.2 PROBLEM SOLVING

LINDO (Schrage, 1987) is used to solve the mathematical program. The procedure for use of LINDO to find an optimal solution has been completely automated. By the use of a single command, LINDO is invoked to read the mathematical program, find the optimal solution and store the non-zero solutions. Provision has been made to solve the problem automatically in the batch mode.

6.3 VERIFICATION

Verification has been carried out at every stage of code development to ensure that the programs worked as intended. A thorough description of the steps used in verification of the program LPGN5.FOR is presented. For other computer programs meticulous verification has been carried out to ensure that they performed as intended.

Verification involves checking to see that computer programs have worked as intended and that there are no errors. LPGN5.FOR is the most important computer program as it generates the linear integer program. In order to verify that LPGN5.FOR works as intended, the program is generated in natural algebraic format instead of the

MPSX format. Two problems were used for verification. The first problem had 130 variables and 143 constraints. The second problem had 722 variables and 745 constraints. For each of the above problems the linear integer program was generated. Verification was carried out in three stages as described below.

Stage 1

The initial program was checked for accuracy. Coefficients in the objective function were checked against faculty inclinations in FI.DAT and found to be as expected. Each of the constraints written were checked and found to be as expected. Variables eliminated were checked against faculty non-availability data and teaching schedules and vice-versa.

Stage 2

Changes were introduced in the scheduling environment and the revised linear integer program was generated. The changes were found to be as expected.

Stage 3

The final schedule was verified to obey all constraints and was found reasonable. In particular we looked at the final schedule and verified that:

- a) The required number of faculty has been assigned to each session;
- b) Commoncode sessions have been assigned the same instructor;
- c) Faculty have not been assigned to more than 3 sessions in one day unless the mastergrid had specified 4 or more commoncode sessions in one day;
- d) Faculty have not been assigned to sessions that are to be conducted at the same time;
- e) Limits on the total number of assignments that may be made to a faculty have been respected.

Similar meticulous steps were taken to carry out verification of the other computer programs to ensure that they performed as intended.

6.4 SAMPLE PROGRAM

A sample program for a 2 instructor - 4 session scheduling problem is shown below with appropriate assumptions.

$$\begin{aligned} \text{MAXIMIZE } & 10 \text{ XA001} + 10 \text{ XB001} + 10 \text{ XB002} \\ & + 10 \text{ XA002} + 10 \text{ XA003} + 10 \text{ XB003} \\ & + 20 \text{ XB004} + 40 \text{ XA004} \end{aligned}$$

SUBJECT TO

$$(a) \text{ XA001} + \text{XB001} = 1$$

(Number of faculty required in session 1 is one)

$$(b) \text{ XTA} - \text{XA001} - \text{XA002} - \text{XA003} - \text{XA004} = 0$$

(Total number of sessions assigned to faculty A)

$$(c) \text{ XTA} \leq 3$$

$$\text{XTA} \geq 1$$

(Upper and lower limits on XTA)

$$(d) \text{ XA001} - \text{XA002} = 0$$

(Common code constraint)

$$(e) \text{ XA001} + \text{XA002} + \text{XA003} + \text{XA004} \leq 4$$

(maximum 4 sessions this week)

$$(f) \text{ XA001} + \text{XA002} + \text{XA003} + \text{XA004} \leq 3$$

(maximum 3 sessions this day)

$$(g) \text{ XA001} + \text{XA004} \leq 1$$

(TCS constraint)

$$(h) \text{ Integer } 8$$

6.5 DATA PROCESSING

In this section we discuss input data processing and output data processing. From the point of view of the user, input of data and output of useful reports/statements is the most important aspect of a computer based system. The user communicates with the system through data input. The system provides useful product to the user mainly through the reports/statements that it generates. Success in the area of input and output data processing largely determines acceptability of the system by the user.

Input of data is in natural spreadsheet format to which the user is accustomed. Data files are compatible with existing methods. No additional data collection or data reporting requirement has been imposed on any of the participants. Two principles for design of data files are identified. The non-zero solutions from LINDO are converted to a final schedule in natural grid format. Provision is made for generating statements of teaching assignments, financial statements and summary of schedule.

6.5.1 Input Data Processing

In this subsection we discuss the various data files that were found necessary for input of data. The approach taken for finding objective function coefficients is different from other researchers. Judgement and experience of the scheduler about the suitability of a faculty for a program is combined with faculty preferences to arrive at a composite benefit for the Institute. This composite benefit is used to generate the objective function coefficients of the mathematical program.

Two principles for design of data files for a DSS are identified. It was realized that implementation efforts will face less resistance if changes suggested were kept to the minimum. Further we realized that a DSS designed by us should not add to the workload of the scheduler or the instructors in terms of the data to be collected and

analyzed. It is our claim that the proposed DSS does not add to data collection requirements at all. Further, it is also a strength of this DSS that it utilizes the same formats for data collection that were being used by the scheduler. In our opinion, these are two principles that need to be borne in mind by designers of decision support systems. While we make no claim about originality in these two ideas, we would like to record our belief that we have practiced these two principles to the letter and that we have not seen a record of adherence to these two principles in the DSS literature.

Since these two principles are, in our opinion, important for design of decision support systems, we give below a statement of the two design principles:

1. The proposed new computer based DSS should not add to the workload of the decision maker or other data providing participants unless it is absolutely necessary.
2. The proposed new computer based DSS should not change the formats used previously for data collection unless it is absolutely necessary.

Data about program contents, course offerings in a planning period and faculty are the three different kinds of data needed for instructor scheduling.

6.5.1.1 Program Contents Data

The particulars of different sessions in different programs in different weeks is recorded in master grids that are stored in separate files under distinct names. A particularly important consideration in designing the master grids has been the need to have the input of data in a natural format. The spreadsheet format is presently used by the scheduler. This format has the inherent advantage of being extremely easy for the human user to read and comprehend. Hence it is this spreadsheet format that has been used for input of data.

The data stored in the master grid consists of session timing, session title, commoncode and the number of faculty required to teach this session. Sessions in the

same master grid with the same common code are required to be taught by the same instructor.

6.5.1.2 Course Offerings Data

A primary data file indicating the number of weeks in the planning period, the starting date of the planning period and the calendar year of the first week of the planning period is set up in the data file WD.DAT.

Provision has been made to recognize a leap year and account for it.

Another primary data file called WD1.DAT is set up to record the number of programs running concurrently in a given week of the planning period.

The particular programs that are offered in different weeks are identified in terms of the specific identification numbers of their respective mastergrids in the file ZFN.DAT and this information is recorded in the data file WD2.DAT.

6.5.1.2.1 Program IP.FOR

A Fortran program IP.FOR is a preprocessor that processes the data from the appropriate master grid files to generate three data files WEEKS.DAT, WDATA.DAT and DATE.DAT.

6.5.1.2.2 WEEKS.DAT

This data file is a composite file containing information about all the weekly offerings in a planning period arranged in a time ordered manner. Thus this data file contains all the sessions for which faculty assignments have to be made.

6.5.1.2.3 WDATA.DAT

This data file summarizes information regarding the total number of weeks in the planning period and the number of programs running concurrently in the different weeks along with the starting and ending dates of these weeks.

6.5.1.2.4 DATE.DAT

This data file records the dates of the days in the planning period.

6.5.1.2.5 PROGRAM CPTD.FOR

The Fortran program CPTD.FOR is a preprocessor that prepares a summary of relevant information about the different sessions in the data files PT.DAT and CD.DAT.

6.5.1.2.6 Data File PT.DAT

The data file PT.DAT contains the date, starting time, ending time, number of faculty required, commoncode, unique session code and unique session number for each session in the planning period for which instructor(s) need to be assigned.

6.5.1.2.7 Data File CD.DAT

The data file CD.DAT records the number of sessions in week I, program J and day K.

PT.DAT and CD.DAT are set up to avoid reading and collating data from the large data file WEEKS.DAT in every run of the subsequent processors.

6.5.1.3 Data About Instructors

There are many kinds of information that are needed before an instructor can be assigned to a session.

6.5.1.3.1 Data File FD.DAT

This data file is used to record the following essential particulars about each different instructor:

- a) Name of the Instructor;
- b) Initials of the Instructor;
- c) Lower and Upper limits on the total number of sessions that an instructor may be assigned to teach.

6.5.1.3.2 Faculty Non-Availability Data

Three essential data files FACNABLOCK.DAT, FACNAFULL.DAT and FACNAPART.DAT record three different types of non-availability of the instructors.

6.5.1.3.3 Data File FACNABLOCK.DAT

This data file records non-availability of an instructor for blocks of days (more than or equal to two days) during which the instructor is not available to teach Institute sessions.

6.5.1.3.4 Data File FACNAFULL.DAT

This data file records non-availability of an instructor for days on which the said instructor is not available for the full day.

6.5.1.3.5 Data File FACNAPART.DAT

This data file records non-availability of an instructor for days when the said instructor is not available for part of the day.

6.5.1.3.6 Data File FTS.DAT

The teaching schedules of the instructors is recorded in this data file in natural format. For example an entry may read " TR 0800-0900 " to indicate that this instructor will be teaching classes from 8 am to 9 am on Tuesdays and Thursdays. Provision has been made for summer semesters when start and end dates may be different for different instructors.

6.5.1.3.7 Data File CS.DAT

There are certain sessions which in the opinion of the scheduler is best taught by a particular instructor. These sessions are termed committed sessions and are to be assigned to the designated instructor. Data pertaining to the committed sessions are recorded in CS.DAT.

6.5.1.3.8 Program FDP.FOR

The program FDP.FOR generates a data file called FI.DAT.

6.5.1.3.9 Data File FI.DAT

The data file FI.DAT records instructor inclinations for teaching courses in the different program offerings on the basis of feedback received from the instructors. The scheduler also uses experience and judgement to assign benefits to the various instructor-program combinations in order to influence assignment of instructors to different programs. We believe that this approach is markedly different from

approaches used by others in the past. Specifically, previous researchers have used personnel preferences to form the objective function coefficient. Our approach uses the same preferences for forming the objective function coefficient but at the same time gives the scheduler the ability to draw on experience and influence assignments by adjusting the benefit points. This may also be used to prohibit particular assignments.

6.5.2 Output Data Processing

In this sub-section we discuss how the solution from LINDO is converted to a final schedule in natural grid format. We also discuss how a statement of teaching assignments of an individual instructor and a financial statement for payment purposes are to be generated.

6.5.2.1 Final Schedule

A Fortran program OP.FOR reads the non-zero variables from data file S.DAT, the program schedules from WEEKS.DAT and generates the final schedule in a data file FINAL.DAT. This schedule is in natural spread sheet format and shows the initials of the instructors assigned to the different sessions in the various programs. The output is in a form that may be read by the Instructors/ participants without further help from the scheduler (Figure 6.1).

6.5.2.2 Statement of teaching assignments

A Fortran program FS.FOR reads the final schedule stored in FINAL.DAT and prepares a comprehensive statement showing detailed particulars of sessions that have been assigned to a particular instructor in a particular planning period. This is done well in advance of the start of a planning period to enable an instructor to plan around the assigned sessions.

FINAL SCHEDULE

WEEK NUMBER : 2

PROGRAM NUMBER : 1

PROGRAM TITLE : WEEK 1 OF THREE WEEK INSTITUTE FOR
PRODUCTIVITY THROUGH QUALITY

DATES : 0108 TO 0114

LOCATION : SMC

SUNDAY	MONDAY	TUESDAY	WEDDAY	THUDAY	FRIDAY	SATDAY
0108	0109	0110	0111	0112	0113	0114
1530-1630 MATH REFRESHER LH	0810-0940 INTRO. TO STAT MGT. KK	0810-0940 PROCESS ANALYSIS JLS	0810-0940 U AND C CHARTS DS	0810-0940 INTRO. TO VARIABLES HA	0810-0940 XBAR & R CHARTS RAM	0810-0940 MOVING AVG AND RANGE CHARTS JRE
1800-1845 RECEPTION	1000-1130 CHARTS FOR FRACTION NONCONFOR KK	1000-1130 PROCESS ANALYSIS JLS	1000-1130 U AND C CHARTS DS	1000-1130 VARIABLES CONTD. R CHARTS KG	1000-1130 XBAR & R CHARTS RAM	1000-1130 MOVING AVG AND RANGE CHARTS JRE
1845-2045 DINNER AND ORIENTN	1300-1430 P CHARTS & RUNS TESTS JLS	1300-1430 PROCESS ANALYSIS JLS	1300-1430 U AND C CHARTS JP	1300-1430 DISTR. OF MEASURE JLS	1300-1430 PROCESS CAPABILITY ANALYSIS JRE	
	1445-1615 np-CHARTS NUMBER NONCONFORM JLS	1445-1615 p-CHARTS [VARYING SAMPLE SIZE] DS	1445-1615 U AND C CHARTS JP	1445-1615 THE NORMAL DISTRIBN. JLS	1445-1615 PROCESS CAPABILITY ANALYSIS JRE	
0700-0750 [MONDAY] COMPUTER LH	1630-1830 P-CHART & PROBLEM SOLVING TOOLS JLS	1630-1830 HILTON CASE STUDY HA, KK, DS	1630-1830 OPEN SESSION	1630-1830 HILTON CONTROL LIMITS KK	1630-1830 CONTROL CHARTS FOR INDIVIDUAL HA	

Figure 6.1. Final Schedule in output file FINAL.DAT.

A facility is provided to generate the statement for any period of shorter duration (Figure 6.2).

6.5.2.3 Financial Statement

A Fortran program FSMT.FOR generates a financial statement showing the number of sessions taught by each instructor in a particular week of interest . The financial statement is sent to the Accounts Department for use in making payments to the instructors. Since sessions of different programs are paid for at different rates, this statement must be generated for each program in the particular week of interest (Figure 6.3).

6.5.2.4 Summary Statement

Whenever a schedule is generated or revised by use of the heuristics a summary statement is generated for use of the user (figure 6.4). The summary statement contains information useful for decision making. In particular, the total number of sessions assigned to an instructor, upper and lower limits of the instructor and number of sessions assigned in different weeks to an instructor are displayed.

STATEMENT OF TEACHING ASSIGNMENT

1.NAME OF FACULTY : HAL AIKENS				
NO.	DATE	TIME	SESSION TITLE	CODE
1	0822	0810-0940	INTRO. TO STAT MGT.	TWI1A
2	0823	1830-2030	HILTON CASE STUDY	TWI1A
3	0829	1445-1615	CIGARRETTE CASE STUDY	TWI2A
4	0829	1830-2030	CIGARRETTE CASE STUDY	TWI2A
5	0903	0810-0940	PROBLEMS OF MEASUREMENT	TWI2A
6	0903	1000-1130	PROBLEMS OF MEASUREMENT	TWI2A
7	0906	1645-1830	HILTON CASE STUDY	PG1A
8	0909	1300-1430	HILTON CASE STUDY	PG1A
9	0909	1445-1615	HILTON CASE STUDY	PG1A
10	0920	0810-0940	COST MANAGEMENT	ASD1

Figure 6.2 . Statement of Teaching Assignments

FINANCIAL STATEMENT	
TOTAL NUMBER OF SESSIONS TAUGHT BY DIFFERENT FACULTY IN PROGRAM::	
PROGRAM TITLE: WEEK 1 OF THREE WEEK INSTITUTE FOR PRODUCTIVITYTHROUGH QUALITY	
WEEK : 1	FROM : 0821 TO : 0827 CODE : TWI1A
FACULTY INITIAL	TOTAL NUMBER OF SESSIONS TAUGHT
HA	2
CC	3
KG	0
LH	1
KK	3
ML	0
RAM	3
JP	6
JRE	3
RS	3

Figure 6.3 Financial Statement for week 0821 to 0827

SUMMARY STATISTICS

SUMMARY OF FACULTY ASSIGNMENTS

INITIALS	TOTAL ASSIGNED	REQUESTS	
		LOW	HIGH
HA	53	50	70
CC	12	5	15
KG	30	30	40
LH	31	25	45
KK	112	100	130
ML	30	30	50
RAM	36	32	52
JP	31	25	40
JRE	35	35	50
RS	67	60	80
JLS	50	40	75
JRS	10	5	20
DS	45	35	55

SUMMARY OF ASSIGNMENTS IN DIFFERENT WEEKS

WEEK NO.	FACULTY INITIAL												
	HA	CC	KG	LH	KK	ML	RAM	JP	JRE	RS	JLS	JRS	DS
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	4	0	3	3	10	1	3	2	5	5	13	0	3
3	2	0	0	0	0	0	8	0	2	15	0	0	2
4	2	2	8	2	7	2	2	6	2	0	0	0	3
5	2	0	4	0	14	4	0	5	5	9	4	0	4
6	10	4	3	0	15	3	2	3	4	2	12	1	8
7	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	7	0	0	12	0	0	2
9	5	0	1	0	15	3	2	3	1	4	6	2	4
10	6	2	3	9	10	4	3	3	2	6	4	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	8	6	1	4	0	2	7	4	0	2
13	0	0	0	0	0	0	0	0	0	0	0	0	0
14	8	0	2	0	11	6	3	7	4	0	3	3	5
15	5	2	5	0	12	4	0	0	3	2	1	0	6
16	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0
19	9	2	1	9	12	2	2	2	5	5	3	2	6

Figure 6.4 Summary Statistics in data file SUMMARY.DAT

CHAPTER VII

CONCLUSIONS AND FUTURE RESEARCH

In this chapter we summarize the highlights of this research and identify promising directions for future work.

7.1 CONCLUSIONS

Training programs are conducted on an ongoing basis by universities and other specialized institutes as part of their effort to impart and update management and technical skills of executives. Each program addresses a particular training need and consists of a set of sessions taught by faculty from different departments in a university and practicing managers from industry. A difficult scheduling problem arises when instructors have to be assigned to sessions of these programs in advance for a typical planning period of four to six months while meeting many complicating requirements.

In practice this problem is solved manually for time periods ranging from two to four weeks and usually results in non-optimal schedules. The focus of this dissertation research is development of optimal and near optimal solutions that provide comprehensive scheduling support to a decision maker. The optimal approach is based on a linear integer programming formulation of the problem which is solved by a commercial solver.

The optimal approach takes over 40 minutes of CPU time. In order to generate a reasonably good quality solution in a short time we have developed and implemented knowledge based heuristics that take less than 25 seconds to generate a

schedule. These heuristics are unique in the sense that they act like junior officials in an organization and construct a preliminary schedule on the basis of prespecified rules. The schedule is presented to the user for review who draws on past experience to issue directions for improving the proposed solution. This process iterates until the scheduler is satisfied. The underlying mechanism of this system closely resembles real life decision making processes in a hierarchical organization and thus departs from the usual computer based expert systems that take decision making away from the decision maker.

We have investigated the quality of the knowledge based heuristic solution by developing upper bounds based on lagrangian relaxation of the scheduling problem. The relaxation uses a subproblem which can be solved efficiently as a minimum cost network flow problem. We have analyzed this subproblem and identified the general characteristics that make it transformable to a network flow problem. Based on these general characteristics we defined the "SARS" model and exploited its structure to develop procedures to solve the problem as a linear program, a minimum cost network flow problem or a uncapacitated transportation problem. We have shown that the Assignment Problem, the Bounded Interval Semi-Assignment Problem and a special case of the Capacitated Transportation Problem with unit upper bounds on flows are special cases of the SARS model.

We have extended the lagrangian relaxation based upper bounding procedures to develop heuristics that construct near optimal solutions in reasonable time.

Our efforts have culminated into the design, development, testing and implementation of a comprehensive decision support system. The three different problem solving approaches based on use of commercial solver, lagrangian heuristic and knowledge based heuristic have been integrated into a hierarchical strategy for

supporting the needs of the user. To ensure minimal resistance in implementation, we have identified and used two principles for design of a decision support system. These principles suggest use of existing data collecting formats and maintaining data reporting volume at existing levels.

7.2 FUTURE RESEARCH TOPICS

In this section we discuss directions for future research. The algebraic structure of the SARS problem is such that it can be transformed into a problem that has a totally unimodular constraint matrix. It would be interesting to investigate real world problems that may contain the SARS problem as a subproblem.

SARS is an integer linear program that may also be solved as a minimum cost network flow problem. It may be possible to investigate and identify other integer programs that may be formulated as a network flow problem.

The knowledge-based heuristic is found to perform quite well on real problems and random problems when the measure of performance is based on some kind of "cost" or "utility". However, not all objectives can be expressed in terms of a "cost" or "utility". For example, it may be undesirable to allow two "low-key" instructors to teach sessions one after another. We would like to investigate how well the knowledge-based heuristic does in meeting such objectives.

In this research we have implemented a mathematical programming approach, a knowledge-based heuristic approach and a lagrangian heuristic approach for the Instructor Scheduling Problem (ISP). A fourth approach involves an "expert" system. If this is implemented, it would be possible to carry out a comparative evaluation of

the four approaches and develop guidelines about suitability of the different approaches for different classes of problems.

BIBLIOGRAPHY

BIBLIOGRAPHY

- Abernathy, W. J. , Baloff, N. , Hershey, J. C., Wandel, S. , " A Three-Stage Manpower Planning and Scheduling Model - A service-Sector Example, " Operations Research, **21** 3, 693 -711 (1973).
- Ardekani, M. B., Mahmood, M. A., " Development and Validation of a Tool For Assigning Students To Groups For Class Projects, " Decision Sciences, **17** 1, 92 -11 (Winter 1986).
- Balinski, M. L. , " Signature Methods for the Assignment Problem," Operations Research, **33**, 527 - 537 (1985).
- Barr, R. , Glover, F. and Klingman, D. , " The Alternating Path Basis Algorithm for Assignment Problems , " Mathematical Programming, **13**, 1 - 13 (1977).
- Bell, P. C., Hay, G. , Liang, Y. , " A Visual Interactive Decision Support System For Workforce(Nurse) Scheduling," INFOR, **24** 2, 134 - 145 (May 1986).
- Bell, P. C., " Visual Interactive Modeling as an Operations Research Technique, " INTERFACES, **15** 4, 26 -33 (July-August 1985).
- Bell, P. C., " Visual Interactive Modelling in Operational Research: Successes and Opportunities, " Journal Of Operations Research Society, **36** 11, 975 -982(1985).
- Bharath, R., " Logic Programming: A Tool For MS/OR ? , " INTERFACES, **16** 5, 80-91 (September-October 1986).
- Bradley, G. H., Graves, G. W., GNET: A Primal Network Code for Solution of Capacitated Network Problems, Knoxville, The University of Tennessee Computing Center, 1975.
- Brearley, A. L., Mitra, G., Williams, H. P., " Analysis of Mathematical Programming Problems Prior to Applying the Simplex Algorithm, " Mathematical Programming, **8** 1, 54 - 83(February 1975).
- Charnes, A., Cooper, W. W., " Management Models and Industrial Applications of Linear Programming," Management Science, **4** 38-91(October 1957).
- Crowder, H. , Johnson, E. L. , Padberg, M. , " Solving Large-Scale Zero-One Linear Programming Problems," Operations Research, **31** 5, 803-834 (September - October 1983).
- Fisher, M. L., " The Lagrangian Relaxation Method for Solving Integer Programming Problems," Management Science, **27** 1, 1 - 18 (January 1981).
- Ford, L. R. , Fulkerson, D. R. , " A Primal Dual Algorithm for the Capacitated Hitchcock Problem," Naval Research Logistics Quarterly, **4**, 47-54 (March 1957).

- Frieze, A. M., Yadegar, J., "An Algorithm for Solving 3-Dimensional Assignment Problems with Applications to Scheduling a Teaching Practice," Journal of the Operational Research Society, 32 11, 989 - 995(1981).
- Fulkerson, D. R. , " An Out-of-Kilter Method for Minimal Cost Flow Problems , " SIAM Journal on Applied Mathematics, 9, 18 - 27 (1961).
- Geoffrion, A., McBride, R., " Lagrangian Relaxation Applied to Capacitated Facility Location Problems," AIIE Transactions, 10 1, 40 - 47(March 1978).
- Gilbert, K. C. , Hofstra, R.B.," Multidimensional Assignment Problems, " Decision Sciences, 19 2, 306-321 (Spring 1988).
- Glasse, C. R. , Mizrach, M. , "A Decision Support System for Assigning Classes to Rooms, " INTERFACES , 16 5, 92 - 100 (September - October 1986).
- Goffin, J. L., " On Convergence Rates of Subgradient Optimization Methods," Mathematical Programming, 13, 329 - 347(1977).
- Goslar, M. D. , Green, G. I. , Hughes, T. H. , " Decision Support Systems: An Empirical Assessment For Decision Making, " Decision Sciences, 17 1, 79-91(Winter 1986).
- Held, M., Wolfe, P., Crowder, H. P., " Validation of Subgradient Optimization," Mathematical Programming, 6, 62 - 88 (1974).
- Henderson, J. C., " Finding Synergy Between Decision Support Systems and Expert Systems Research," Decision Sciences 18 3, 333 - 349 (Summer 1987).
- Hill, A. V., Naumann, J. D., Chervany, N. L. , " SCAT and SPAT: Large Scale Computer-Based Optimization Systems for the Assignment Problem, " Decision Sciences, 14 2, 207 - 219 (April 1983).
- Hofstra, R. M. B. , " Multidimensional Assignment and Scheduling with Application to a Tourism Convention Scheduling Problem, " Ph.D Thesis, University of Tennessee, Management Science Program, Knoxville, TN(1983).
- Hung, M. S. , " A Polynomial Simplex Method for the Assignment Problem, " Operations Research, 31, 595 - 600 (1983).
- Johnson, E. L. , Kostreva, M. M. , Suhl, U. H. , " Solving 0-1 Integer Programming Problems arising from Large Scale Planning Models, " Operations Research, 33 4, 803 -819 (July-August 1985).
- Jornsten, K., Nasberg, M., " A New Lagrangian Relaxation Approach to the Generalized Assignment Problem," European Journal of Operational Research, 27, 313 - 323 (1986).
- Klingman, D. , Randolph, P. H. , Fuller ,S. W. , " A Cotton Ginning Problem," Operations Research , 24 4, 700-717 (July-August 1976).

- Kolesar, P. J. , Rider, K. L. , Crabill, T. B. , Walker, W.E., " A Queueing-Linear Programming Approach to Scheduling Police Patrol Cars, " Operations Research, 23 6, 1045 - 1062 (November- December 1975).
- Kuhn, H. W. , " The Hungarian Method for the Assignment Problem, " Naval Research Logistics Quarterly, 2, 83 - 97 (1955).
- Lawler, E. L. , Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York (1976).
- Lawler, E. L., Wood, D. E., " Branch-And-Bound Methods: A Survey," Operations Research, 14, 699-719.
- Liang, T. T. , Thompson ,T. J. , " A Large-Scale Personnel Assignment Model for the Navy, " Decision Sciences, 18 2, 234-249 (Spring 1987).
- Maier-Rothe, C., Wolfe, H. B., " Cyclical Scheduling and Allocation of Nursing Staff, " Socio-Economic Planning Sciences, 7, 471 - 487 (1973).
- Magazine, M. J. , " Scheduling A Patient Transportation Service in a Hospital, " INFOR, 15 2, 242 - 254 (June 1977).
- McClure, R.H. , Wells, C. E., " A Mathematical Programming Model for Faculty Course Assignments, " Decision Sciences, 15 3, 421 - 433 (Summer 1984).
- Megeath, J. D. , " Successful Hospital Personnel Scheduling," INTERFACES, 8 2, 55 - 59 (February 1978).
- Miller, H. E. , Pierskalla ,W. P. , Rath, G. J., "Nurse Scheduling Using Mathematical Programming, " Operations Research, 24 5, 857-870 (September - October 1976).
- Miller, H. E., " Personnel Scheduling in Public Systems: A Survey, " Socio-Economic Planning Sciences, 10, 241-249(1976).
- Muckstadt, J. A., Koenig, S. A., " An Application of Lagrangian Relaxation to Scheduling in Power - Generation Systems," Operations Research, 25 3, 387 - 403 (May - June,1977).
- Nemhauser, G. L. , Wolsey, L. A. Integer and Combinatorial Optimization. First Edition. New York: John Wiley & Sons, 1988.
- Ow, P. S. , Smith, S. F. , " Two Design Principles for Knowledge-Based Systems, " Decision Sciences , 18 3, 430 - 447 (Summer 1987).
- Plotnicki, W. J. , Garfinkel, R. S., " Scheduling Academic Courses to Maximize Student Flow: A Simulation Approach," Socio-Economic Planning Sciences, 20 4, 193-199(1986).

- Rosenthal, R. E. , " Principles of Multiobjective Optimization, " Decision Sciences, 16 1, 133-152 (Winter 1985).
- Ross, G. T. , Soland, R. M. , " A Branch and Bound Algorithm for the Generalized Assignment Problem," Mathematical Programming 8, 91-103 (1975).
- Ross, G. T. , Soland , R. M. , Zoltners , A. A. , " The Bounded Interval Generalized Assignment Model," Naval Research Logistics Quarterly , 27 4, 625-633 (December 1980).
- Schrage, L. User's Manual for Linear Integer and Quadratic Programming with LINDO .Third Edition,Red Wood City, California: Scientific Press, 1987.
- Segal, M., " The Operator-Scheduling Problem: A Network-Flow Approach," Operations Research, 22 4, 808 - 823 (July - August 1974).
- Sullivan, R.S. , Secrest, S.C., " A Simple Optimization based DSS for Production Planning at Dairyman's Cooperative Creamery Association," INTERFACES, 15 5, 46 -53 (September-October 1985).
- VAXcluster User's Guide.Knoxville. The University of Tennessee Computing Center, 1986.
- Wagner, H.M., " On A Class Of Capacitated Transportation Problems," Management Science , 5, 304-318 (1959).
- Warner, D. M. , " Scheduling Nursing Personnel According to Nursing Preference: A Mathematical Programming Approach,"Operations Research, 24 5, 842-856 (September-October 1976).
- Weiss, S. M. , Kulikowski, C. A., A Practical Guide to Designing Expert Systems, First Edition, New Jersey:Rowman and Allenheld, 1984.
- Williams, H. P. , "The Reformulation of Two Mixed Integer Programming Problems," Mathematical Programming, 14, 325 - 331 (1978).

VITA

Arup Kumar Mukherjee was born in Calcutta, India on December 2, 1953. He attended Loyola School in Jamshedpur and graduated in the year 1971. He earned a Bachelor of Technology degree in Electrical Engineering from the Indian Institute of Technology, Kanpur, India in April, 1977.

He came to the United States in September, 1986. He will receive the Doctor of Philosophy degree at the University of Tennessee, Knoxville in 1991 with a major in Management Science.

He is a member of The Operations Research Society of America, The Decision Sciences Institute, The Production and Operations Management Society and The Institution of Engineers(India). He is a Certified Associate of The Indian Institute of Bankers and Chartered Engineer(India).

The author has held industry positions in marketing, banking, sick industry rehabilitation and industrial finance.

He has joined the faculty of the Computer Information Systems Department at The Fort Hays State University in August 1990.