This work is protected by copyright and other intellectual property rights and duplication or sale of all or part is not permitted, except that material may be duplicated by you for research, private study, criticism/review or educational purposes. Electronic or print copies are for your own personal, non-commercial use and shall not be passed to any other individual. No quotation may be published without proper acknowledgement. For any other use, or to quote extensively from the work, permission must be obtained from the copyright holder/s.

SCHEDULING, USING BRANCH AND BOUND TECHNIQUES

A thesis submitted for the degree
of Doctor of Philosophy at
the University of Keele

by

AHMED MOHAMED AHMED HARIRI

Department of Mathematics,
University of Keele.
December 1981.

بسم الله الرحمن الرحيم

IN THE NAME OF ALLAH THE COMPASSIONATE THE MERCIFUL

TO MY PARENTS
AND MY WIFE

PREFACE

This thesis presents the results of research carried out by the author at the University of Keele, 1978-1981. Except where acknowledged otherwise, the work reported here is claimed as original, and has not previously been submitted for a higher degree at this or any other university.

I wish to thank Dr. Chris Potts, my supervisor, for his wonderful guidance and help throughout; King Abdul-Aziz University, Jeddah, Saudi Arabia for their financial support; and members of the Computer Centre, especially Dr. A. F. Grundy and Dr. P. G. Collis for their valuable comments regarding the computer programs and Mrs. C. Parnell for her excellent typing of the thesis.

Finally, I would like to express my utmost appreciation to my parents for their unfailing encouragement and support during my studies at school and university, and my wife for her patience and for providing badly needed moral support.

ABSTRACT

The thesis is devoted to machine scheduling problems. It is presented in four parts.

Part I is an introductory one in which we give a full description of machine scheduling problems together with existing methods of approach to solving these problems.

Part II is started by giving a review of one machine problems together with well known and new heuristics for most of these problems. Then we use branch and bound techniques to solve a one machine problem with release dates to minimize the sum of weighted completion times "i.e. the $1/r_i/\Sigma w_i C_i$ problem" and a one machine problem to minimize the weighted sum of squares of completion times "i.e. the $1//\Sigma w_i C_i^2$ problem".

We start Part III by giving a review of methods to solve the flow- and job-shop scheduling problems. We then apply branch and bound techniques to solve the two and m machine permutation flow-shop problems with precedence constraints to minimize the maximum completion time in each case "i.e. the $F2/prec/C_{max}$ and $Pm/prec/C_{max}$ problems".

Part IV contains our conclusion together with a brief look at multi-processor computers and their impact on the future of scheduling.

CONTENTS

		Page
PART I:	INTRODUCTION	1
CHAPTER 1:	INTRODUCTION	2
1.1	Introduction	2
1.2	Historical Background	2
1.3	Contributions of this Research	3
CHAPTER 2:	DESCRIPTION OF MACHINE SCHEDULING PROBLEMS	6
2.1	Introduction	6
2.2	Restrictions	6
2.2.1	Restrictions on the Machines	6
2.2.2	Restrictions on the Jobs	7
2.3	Objectives	11
2.4	Problems Classification	14
2.4.1	Machine Environment	14
2 4.2	Job Characteristics	15
2.4.3	Objective Function	16
2.4.4	Examples	16
2.5	Computational Complexity	16
CHAPTER 3:	METHODS OF APPROACH	18
3.1	Introduction	18
3.2	Branch and Bound Approach	21
3.2.1	The Bounding Procedure	22
3.2.2	The Branching Procedure	24
3.2.3	The Search Strategy	24
3.3	Heuristic Methods	26
3.3.1	Sampling Technique	27
3.3.2	Priority Rules	28
3.3.3	The Tree Type Heuristic	30
PART II:	SINGLE MACHINE SCHEDULING	32
CHAPTER 4:	SINGLE MACHINE SCHEDULING	33
4.1	Introduction	33
4.2	Minimizing Maximum Cost f_{max}	33
4.3	Minimizing Total Cost Σf_i	35
4.3.1	$1/\beta/\Sigma w_i C_i$	35
4.3.2	$1/\beta/\Sigma w_i C_i^2$	36
4.3.3	$1/\beta/\Sigma w_i T_i$	37
4.3.4	$1/\beta/\Sigma w_i U_i$	38
4.4	Multiple Objectives	39

Page

CHAPTER 5: HEURISTICS FOR SINGLE MACHINE PROBLEMS 43

5.1 Introduction 43
5.2 Heuristics Chosen from the Literature 44
5.2.1 $1/r_i/L_{max}$ 44
5.2.2 $1/r_i/\Sigma C_i$ 46
5.2.3 $1/d_i/\Sigma w_i C_i$ 47
5.2.4 $1/prec/\Sigma w_i C_i$ 48
5.2.5 $1//\Sigma T_i$ 50
5.3 New Heuristics 52
5.3.1 $1/r_i,prec/\Sigma w_i C_i$ 53
5.3.2 $1/r_i,prec/\Sigma w_i C_i^2$ 54
5.3.3 $1/r_i,prec/\Sigma w_i T_i$ 54
5.3.4 $1/r_i,prec/\Sigma w_i U_i$ 56
5.4 The Tree Type Heuristic 60
5.4.1 The Algorithm 60
5.4.2 Computational Experience 61
5.4.2.1 Test Problems 61
5.4.2.2 Computational Results 62
5.5 Concluding Remarks 65

CHAPTER 6: AN ALGORITHM FOR SINGLE MACHINE SEQUENCING WITH RELEASE DATES TO MINIMIZE TOTAL WEIGHTED COMPLETION TIME 66

6.1 Introduction 66
6.2 The Heuristic Method 67
6.3 Derivation of the Lower Bound 69
6.4 The Improved Lower Bound 72
6.5 Dominance Rules 76
6.6 The Algorithm 78
6.7 Modified Algorithm 80
6.7.1 Branching 80
6.7.2 Composite Jobs 85
6.7.3 Distributing the Multiplier of a composite job amongst its component jobs 87
6.7.4 Procedure: Distributing λ_K of a Composite Job K amongst its component jobs 88
6.7.5 Implementation of the Mixed Branching (MB) 89
6.8 Computational Experience 95
6.8.1 Test Problems 95
6.8.2 Computational Results 95
6.9 Concluding Remarks 99

CHAPTER 7: THE SINGLE MACHINE PROBLEM WITH WEIGHTED SUM OF SQUARES OF COMPLETION TIMES 100

7.1 Introduction 100
7.2 Dominance Rules 101
7.3 Townsend Lower Bound 102

Page

7.4 New Bounding Procedure 105
7.5 Implementation of the Lower Bound 115
7.6 Example 119
7.7 The Algorithm 124
7.8 Incorporating the Dominance Rules with the Lower Bound 126
7.9 Precedence Constraints 127
7.10 Computational Experience 130
7.10.1 Test Problems 130
7.10.2 Computational Results 130
7.11 Concluding Remarks 132

PART III: MULTI-MACHINE SCHEDULING 134

CHAPTER 8: FLOW-SHOP SCHEDULING 135

8.1 Introduction 135
8.2 The $Pm//C_{max}$ Problem 138
8.2.1 Branching Rule 138
8.2.2 Lower Bounds 139
8.2.3 Dominance Rules 144
8.2.4 Heuristic Methods 146
8.3 Open- and Job-Shop Problems 150

CHAPTER 9: THE TWO MACHINE FLOW-SHOP PROBLEM UNDER PRECEDENCE CONSTRAINTS 152

9.1 Introduction 152
9.2 Branching Rule and Dominance 154
9.3 Lower Bounds 157
9.3.1 Job Based Bound 158
9.3.2 Conflict Bound 159
9.4 Heuristic 161
9.5 Example 161
9.6 The Algorithm 166
9.7 Computational Experience 167
9.7.1 Algorithm Representation 167
9.7.2 Test Problems 167
9.7.3 Computational Results 168
9.8 Concluding Remarks 176

CHAPTER 10: THE GENERAL PERMUTATION FLOW-SHOP PROBLEM UNDER PRECEDENCE CONSTRAINTS 180

10.1 Introduction 180
10.2 Lower Bound 181
10.3 The Algorithm 184
10.3.1 Branching Rule 184
10.3.2 Lower Bounds 184
10.3.3 Dominance Rules 185
10.3.4 Implementation of the Dominance Rules 188
10.3.5 Upper Bounds 189
10.4 Computational Experience 190
10.4.1 Test Problems 190
10.4.2 Computational Results 191
10.5 Concluding Remarks 198

		Page
PART IV: CONCLUSION		199
CHAPTER 11: CONCLUSION		200
11.1	Contribution of this Research	200
11.2	Future of Scheduling	201
	APPENDIX	
	REFERENCES	

PART I

INTRODUCTION

CHAPTER ONE

INTRODUCTION

1.1 Introduction

This study will be devoted to machine scheduling problems. The problems that will be under our consideration can be defined as follows. There are a given number of jobs each of which requires one or more operations. An operation is the processing of a job on a machine. It is required to determine the starting times of the operations.

Although this definition suggests that the problem is mainly applicable to industrial production, it can be interpreted to cover various other situations: jobs and machines can stand for patients and hospital equipment, ships and dockyards, programmer and computers, etc. Clearly, scheduling algorithms are of much importance to operational research practitioners.

1.2 Historical Background

Scheduling problems have been under study for a long time, but the first break through in scheduling came in 1954 in the form of a paper by Johnson (Johnson, 1954). Two other important results followed shortly (Jackson, 1955; Smith, 1956).

Encouraged by the fact that the simplex method can be used to solve linear programming problems, Bowman (Bowman, 1959) formulated scheduling problems as an integer programming problem, hoping that a good algorithm for solving the latter one could be found. Others followed in the same direction, but they soon abandoned this approach, firstly because of the hundreds of 0-1 variables and constraints required to formulate scheduling problems (even of small sizes), and secondly because no good general algorithm has been found to solve 0-1 programming problems.

Branch and bound techniques were developed and first used by Eastman (Eastman, 1959) for the travelling salesman problem and by Land and Doig (Land and Doig, 1960) in the context of mixed integer programming. They were first applied to scheduling problems by (Ignall & Schrage, 1965; Lomnicki, 1965; Brown & Lomnicki, 1966; McMahon & Burton, 1967).

The difficulty of scheduling problems made heuristic methods (methods that do not guarantee optimality) unavoidable for many problems. Simulations of actual and hypothetical environments were used to test the performance of these heuristics. Unfortunately, not enough work has been done on the worst case behaviour of these heuristics. First results on the worst case performance of heuristics were due to Graham (Graham, 1966, 1969). A review of worst case performance of scheduling heuristics can be found in (Garey, Graham and Johnson, 1978). However, heuristic methods are likely to become a major research area in the near future because of their importance in real life situations.

Classifying scheduling problems according to their degree of algorithmic complexity was first reported in (Cook, 1971) and (Karp, 1972). However, major development in the classification and complexity of scheduling problems is mainly due to Rinnooy Kan (Rinnooy Kan, 1976), Lenstra (Lenstra, 1977) and (Garey & Johnson, 1979).

1.3 Contributions of this research

As mentioned before, this study is devoted to machine scheduling problems. It is presented in four parts.

Part I is an introductory one. We start this part by giving a full description of machine scheduling problems, including notations and representations. We shall distinguish between three types of problems: "easy", "hard" and "open" scheduling problems. This is followed by an extensive discussion of various restrictions which will be assumed (unless stated

otherwise) throughout this study. The most important of these is the restriction to deterministic problems, which eliminates all stochastic aspects like queueing theory, and also the restriction to unit machine capacities. Further to that we will restrict ourselves to choosing specific cost functions as optimality criteria. Here, we will restrict ourselves to the so-called regular measures, i.e. criteria in which the quality of a schedule is a non-decreasing function of the jobs' completion times.

In this thesis, we shall assume that once the processing order of the operations has been determined on each machine, each operation is completed as soon as possible, and hence we do not distinguish between the two concepts "feasible sequences" and "feasible schedules". However, some researchers (Elmaghraby, 1968; Ashour, 1972; Rinnooy Kan, 1976) distinguish between these two concepts: a sequence corresponds to a processing order of operations on each machine, while a schedule determines the exact starting and finishing times of each operation besides determining the processing order of the operations.

We end Part I by listing most well-known methods of approach to solving scheduling problems, e.g. branch and bound, dynamic programming, heuristics, etc. We shall discuss in detail two of these methods, namely heuristic and branch and bound approaches. Heuristic methods are included because of their importance in real life situations. The branch and bound method is included because it is amongst the most widely used methods of approach to solving scheduling problems, and because it is the method to be used throughout this thesis, except in Chapter 5, where we shall give heuristic methods for solving one machine problems.

Part II is devoted to single machine problems. We start this part by giving a brief survey of the principal results which are classified according to the optimality criterion chosen. This is followed by

heuristic methods for solving one machine problems, some of which are chosen from the literature; others are new ones for which some computational experience is reported. We end part II by demonstrating the properties and performance of branch and bound techniques on two single machine problems: single machine sequencing with release dates to minimize total weighted completion time and single machine sequencing to minimize a quadratic function of completion times. Dominance rules and heuristics will also be included. Computational experiences will also be reported in every case.

Part III is devoted to the general flow-shop problems. The flow-, job- and the open-shop problems will be discussed briefly. The special case known as the permutation flow-shop problem will be discussed in detail. Dominance rules, heuristics, branching rules and lower bounds are reviewed.

We end this part by demonstrating the performance of branch and bound algorithms on two problems: the two-machine flow-shop and the permutation flow-shop problems under precedence constraints to minimize the maximum completion time in each case. As usual, computational experience will be included.

Part IV contains our conclusion together with a brief look at multiprocessor computers and their future impact on both "easy" and "hard" scheduling problems.

CHAPTER TWO

DESCRIPTION OF MACHINE SCHEDULING PROBLEMS

2.1 Introduction

Machine scheduling problems can be described as follows. There are n jobs (numbered 1, ..., n) and job i (i=1, ..., n) requires m_i operations. Each operation corresponds to the processing of a job on one of m machines for a given period of time. The problem is to find the optimal processing order of these operations on each machine which minimizes, subject to some constraints, a given objective function.

In Section 2.2 we discuss all restrictions on the machines and the jobs, including the ones we shall drop at some stage of this thesis. Objective functions are discussed in Section 2.3, followed by problem classification in Section 2.4. Finally, in Section 2.5, we discuss the computational complexity of machine scheduling problems.

2.2 Restrictions

2.2.1 Restrictions on the machines

Unless stated otherwise, we will restrict ourselves to the following restrictions on the machines.

M1 *The set of machines is known and fixed.*

M2 *All machines are independent, are available to process jobs at the same time and remain available to process jobs during an unlimited period of time.*

M3 *Each machine k(k=1,...,m) is either waiting to process the next job, operating on a job or having finished its last job.*

M4 *All machines are equally important.*

M5 *Each machine has to process all jobs assigned to it.*

M6 *Each machine can process not more than one job at a time.* This restriction will be relaxed in Chapters 8 and 10 to obtain lower bounds for the m machine problems discussed there.

M7 *The processing order per machine is unknown and has to be fixed.*

We point out that restrictions M1 and M5 distinguish between the deterministic problems which we are interested in from the stochastic ones.

For an introduction to the theory developed for stochastic problems, we refer to (Conway et al., 1967: Chapters 7-10) or any book on queueing theory.

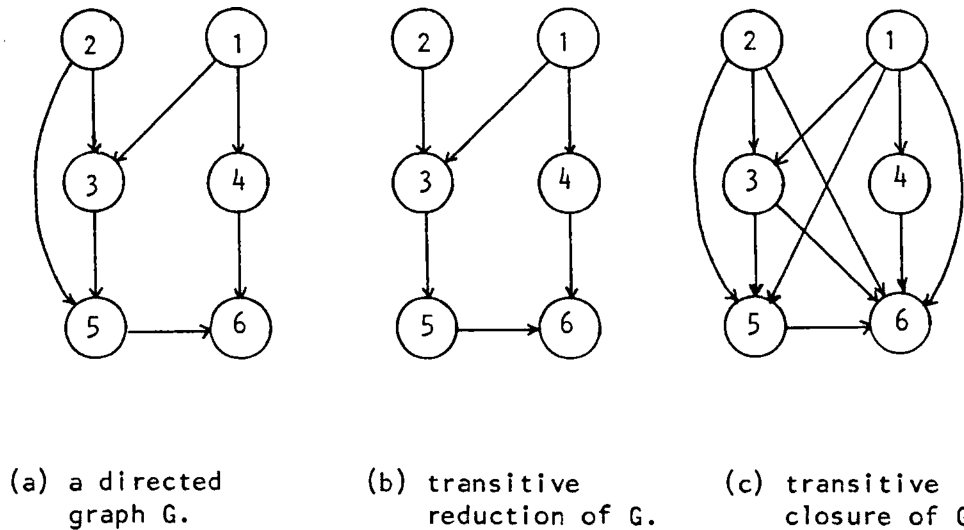
2.2.2 Restrictions on the Jobs

Unless stated otherwise, we will limit ourselves to the following restrictions on the jobs.

J1 *The set of jobs is known and fixed.*

J2 We shall face some situations (see Chapters 4, 5 and 6) where each job $i(i=1,\dots,n)$ has a non-negative integer *release date* r_i at which job i becomes available for processing (we shall use available to denote available for processing). *Unless stated otherwise we shall assume that all the jobs are available at time zero,* i.e. $r_i = 0$ for all $i=1,\dots,n$. We shall also face other situations (see Chapters 4, 5, 9 and 10) where the processing of some of the jobs is dependent on the processing of some other jobs. This situation arises when *precedence constraints* among jobs exist as a part of the problem. These precedence constraints on the jobs *can be represented by a directed acyclic graph* (digraph) $G = (V,E)$, where V denotes the set of vertices and E the set of edges. The vertices of G represent the jobs and the edges represent the arcs between the jobs. *Job i must be processed before job j on each machine if there exists a directed path from vertex i to vertex j in E.* The *transitive closure* of the directed graph G is the graph obtained by adding all arcs (i,j) (if they do not already exist) to G whenever there is a directed path from vertex i to vertex j.

The *transitive reduction* of G is the graph obtained by deleting all arcs (i,j) from G whenever there is a directed path from vertex i to vertex j which does not include the arc (i,j) itself. The *inverse* of G is the graph obtained by reversing the directions of all arcs. The *adjacency matrix* of the precedence constraints is the n x n matrix $X = (x_{ij})$, where $x_{ij} = 1$ if an arc (i,j) exists in the transitive closure of G and $x_{ij} = 0$ otherwise. A precedence graph G, together with its transitive reduction and its transitive closure are given in Figure 2.1.



(a) a directed graph G. (b) transitive reduction of G. (c) transitive closure of G.

Figure 2.1: Precedence constraints as a directed acyclic graph

A precedence graph G = (V,E) is *a tree* if the number of arcs leaving (or entering) a node is at most one. It is called *series-parallel* if G consists of a single node i, i.e. $G = (i,\emptyset)$, or if $G_1 = (V_1,E_1)$ and $G_2 = (V_2,E_2)$ are series-parallel with $V_1 \cap V_2 = \emptyset$ and:

(a) G is the series composition of G_1 and G_2, i.e.

$G = (V_1 \cup V_2, E_1 \cup E_2 \cup (V_1 \times V_2))$, or

(b) G is the parallel composition of G_1 and G_2, i.e.

$G = (V_1 \cup V_2, E_1 \cup E_2)$.

The directed graph G of Figure 2.1 becomes series-parallel if the arc (1,3) is removed from G or if a new arc (1,2), (2,4) or (3,4) [(2,1), (4,2) or (4,3)] is added to G. The simplest acyclic digraph which is not series-parallel is shown in Figure 2.2.

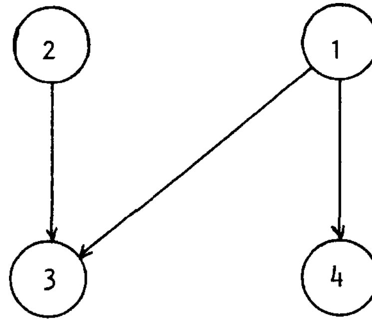


Figure 2.2: A non-series parallel digraph

The precedence graphs which will be considered from now on are of the general (arbitrary) type. More details about series-parallel graphs can be found in (Sidney, 1975; Lawler, 1978; Sidney, 1979; Monma, 1979; Monma and Sidney, 1979; Monma, -).

A different definition of precedence constraints appeared in (Rinnooy Kan, 1976): job i is said to have precedence over job j if it is required that job i is completed on the last machine before job j is started on the first one. Our definition of precedence constraints is more realistic since there appears no reason why job j must be delayed until job i is completed on all the m machines in real life situations. *If precedence constraints are not given as a part of the problem we shall assume that $E = \emptyset$ (i.e. all jobs are independent).*

Also, we shall face some situations (Chapter 4) where each job i (i=1,...,n) must be finished before its deadline d_i (time after which job i will not be available for processing). *Unless stated*

otherwise we shall assume that all jobs remain available during an unlimited time.

J3 *At any instant of time, each job is either waiting for the next machine, being processed by a machine or has been completed processing on its last machine.*

J4 Some of the problems considered in this thesis (see for example Chapters 4, 5, 6 and 7) will have *weights* attached to the jobs to indicate the relative importance of each of these jobs. We shall use w_i to denote the weight assigned to job $i(i=1,\dots,n)$. However, in all other situations we shall assume that all jobs are equally important (i.e. $w_i=1$ for all $i=1,\dots,n$).

J5 *Each job must be processed by all the machines assigned to it.*

J6 *Each job is processed by one machine at a time.* For situations where this restriction is dropped we refer to (Rinnooy Kan, 1976: Section 5.3).

J7 *All processing times are fixed and sequence-independent. Also, there are no setup times.* For situations where there are setup times we refer to (Rinnooy Kan, 1976: Sections 4.2.2 and 4.4.2).

J8 *Each operation once started has to be completed without interruption.* This restriction is relaxed in Section 6.4 where we allow job-splitting (pre-emption) in order to obtain a lower bound for the single machine problem with release dates to minimize the sum of weighted completion times.

J9 *The processing order for each job on the machines is known and fixed.* This restriction is relaxed in Section 8.3 where we talk about the open-shop problem for which this processing order is immaterial.

Restrictions J1 and J5 again stress the deterministic character of the scheduling problems discussed in this thesis.

2.3 Objectives

The aim in all scheduling problems considered in this thesis is to find a schedule that minimizes a given objective function(s). It will be useful at this stage to associate the following data with job $i (i=1,\ldots,n)$:

- *a processing time* p_{ik} of its kth operation, $k=1,\ldots,m_i$ (if $m_i = 1$ for all i, we shall write p_i instead of p_{i1}).
- *a weight* w_i indicating the relative importance of job i. Unless stated otherwise, we assume that $w_i=1$ for all $i=1,\ldots,n$.
- *a release date* r_i (earliest possible starting time for job i). Unless stated otherwise, we assume that $r_i=0$ for all $i=1,\ldots,n$.
- *a due date or deadline* d_i.
- *a cost function* f_i.

We assume that all data (except f_i) to be *non-negative integers*.

Given a processing order on each machine, one can calculate the following (for job $i=1,\ldots,n$):

- the *completion time* of job i denoted by C_i.
- the *lateness* of job i denoted by L_i ($L_i=C_i-d_i$).
- the *tardiness* of job i denoted by T_i ($T_i = \max(0,L_i)$).
- $U_i=0$ if $C_i \leq d_i$ and 1 otherwise.

As in (Rinnooy Kan, 1976) we shall restrict ourselves to *regular measures*, i.e. real functions $f(C_1,\ldots,C_n)$ such that:

$$f(C_1,\ldots,C_n) < f(C_1',\ldots,C_n')$$

implies that $C_i < C_i'$ for at least one job i. These functions usually take one of the following forms:

1. $$f = f_{max} = \max_i\{f_i(C_i)\}$$

2. $$f = \Sigma f_i = \sum_{i=1}^{n} f_i(C_i)$$

Thus we shall seek to minimize either the maximum cost f_{max} or the total cost Σf_i. The following objective functions have frequently been chosen to be minimized:

$$f = \Sigma C_i = \sum_{i=1}^{n} C_i$$ *sum of completion times.*

Introducing weights w_i $(i=1,\ldots,n)$, we have:

$$f = \Sigma w_i C_i = \sum_{i=1}^{n} w_i C_i$$ *weighted sum of completion times.*

$$f = \Sigma w_i C_i^2 = \sum_{i=1}^{n} w_i C_i^2$$ *weighted sum of squares of completion times.*

Introducing due dates $d_i (i=1,\ldots,n)$ we have the following objective functions:

$$f = L_{max} = \max_i \{L_i\}$$ *maximum lateness.*

$$f = T_{max} = \max_i \{T_i\}$$ *maximum tardiness.*

$$f = \Sigma T_i = \sum_{i=1}^{n} T_i$$ *total tardiness.*

$$f = \Sigma U_i = \sum_{i=1}^{n} U_i$$ *total number of late jobs.*

We may also choose to minimize:

$$f = \Sigma w_i T_i = \sum_{i=1}^{n} w_i T_i$$ *weighted sum of tardiness.*

$$f = \Sigma w_i U_i = \sum_{i=1}^{n} w_i U_i$$ *weighted sum of late jobs.*

Let F_i denote the time job i spends in the system (i.e. $F_i = C_i - r_i$) and W_i denotes the waiting time of job i, i.e. $W_i = C_i - (r_i + \sum_{k=1}^{m_i} p_{ik})$.

Rinnooy Kan (Rinnooy Kan, 1976) showed that criteria based on the objective functions: $\Sigma w_i C_i$, $\Sigma w_i F_i$, $\Sigma w_i W$ and $\Sigma w_i L_i$ to be equivalent (i.e. have identical optimum schedules). He also showed that an optimal schedule with respect to L_{max} is also optimal with respect to T_{max}.

Remark: L_{max} and T_{max} are not equivalent: a schedule with $T_{max}=0$ may be suboptimal with respect to L_{max}.

In Figure 2.3, we give a graph copied from (Lawler, Lenstra and Rinnooy Kan, 1981). The graph defines elementary reduction among scheduling problems. An arc from vertex V_1 to vertex V_2 in this graph denotes that problem P_1 is polynomially reducible to problem P_2. It follows that:

- if $P_2 \varepsilon P$ (i.e. polynomially solvable), then $P_1 \varepsilon P$;
- if P_1 is NP-hard, then P_2 is NP-hard.

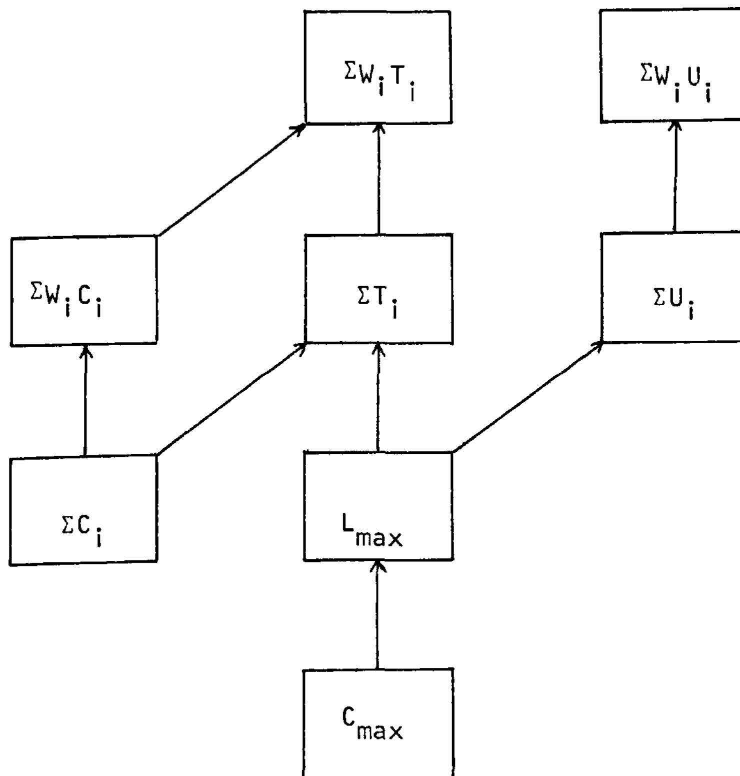


Figure 2.3: Reduction among scheduling problems

2.4 Problems Classification

As we mentioned before, each scheduling problem requires processing n jobs (numbered 1,...,n) on m machines (numbered 1,...,m) so as to minimize some objective function(s). Therefore, each scheduling problem involves well-defined set of jobs, machines and objective function(s). For this reason, scheduling problems are usually described using a 3-parameter notation $\alpha|\beta|\gamma$ to be defined below (Lawler, Lenstra & Rinnooy Kan, 1981).

2.4.1 Machine Environment (α)

The first parameter $\alpha = \alpha_1 \alpha_2$

where $\alpha_1 \varepsilon \{\emptyset, 0, P, F, J\}$. Each of these symbols denotes a specific machine environment ($\emptyset$ is the empty symbol):

$\alpha_1 = \emptyset$: *a single machine problem* ($p_{i1} = p_i$).

$\alpha_1 = 0$: *an open-shop problem* (in which each job i consists of *a set of operations* $\{0_{i1},\ldots,0_{im}\}$. But the order in which the operations are executed is immaterial.

If $\alpha_1 \varepsilon \{P, F, J\}$ *an ordering is imposed* on the set of operations corresponding to each job.

$\alpha_1 = P$: We have *a permutation flow-shop problem,* in which each job has the same sequence of operations. Also, all machines handle the jobs in the same order.

$\alpha_1 = F$: We have *a flow-shop problem,* in which each job has the same sequence of operations, but some job may overtake another job on some machine, i.e., the machines may handle the jobs in different orders.

$\alpha_1 = J$: We have *a job-shop problem,* in which each job has a specified sequence of operations which may differ from the sequence of operations of other jobs.

If α_2 is a positive integer, then m is constant. If, on the other hand, $\alpha_2 = \emptyset$, then m is assumed to be variable. It is obvious that $\alpha_1 = \emptyset$ if, and only if $\alpha_2 = 1$.

2.4.2 Job characteristics

The second parameter $\beta \in \{\beta_1, \ldots, \beta_6\}$ indicates the dropped restrictions by means of the notation given in Section 2.2. A list of the restrictions that we shall drop occasionally will now be given.

1. $\beta_1 \varepsilon \{pmtn, \emptyset\}$

β_1 = pmtn:	Pre-emption (job splitting) is allowed (i.e. dropping restriction J9).
$\beta_1 = \emptyset$:	Pre-emption is not allowed.

2. $\beta_2 \varepsilon \{prec, tree, \emptyset\}$

β_2 = prec:	Precedence relation between jobs are specified which form a precedence graph G of the general (arbitrary) type.
β_2 = tree:	Precedence relations between jobs (i.e. G) form a tree.
$\beta_2 = \emptyset$:	No precedence relations are specified (i.e. jobs are independent).

3. $\beta_3 \varepsilon \{r_i, \emptyset\}$

$\beta_3 = r_i$:	Arbitrarily release dates are specified.
$\beta_3 = \emptyset$:	r_i=0 for all i=1,...,n (i.e. all jobs are available at the same time).

4. $\beta_4 \varepsilon \{m_i \leq \bar{m}, \emptyset\}$

$\beta_4 = m_i \leq \bar{m}$:	A constant upper bound is specified (only if α_1 = J).
$\beta_4 = \emptyset$:	All m_i are arbitrary integers.

5. $\beta_5 \varepsilon \{p=1, p \leq p^*, \emptyset\}$

$\beta_5 = p_{ij} = 1$:	Each operation has unit processing time.
$\beta_5 = p_{ij} \leq p^*$:	Upper bound on all processing times.
$\beta_5 = \emptyset$:	All p_{ij} (p_i) are arbitrary integers.

6. $\beta_6 \varepsilon \{p_i \leq p_j \rightarrow w_i \geq w_j, \emptyset\}$

$\beta_6 = p_i \leq p_j \rightarrow w_i \geq w_j$:	Agreeable weights.
$\beta_6 = \emptyset$:	All w_i are arbitrary integers.

2.4.3 Objective function

The third parameter $\gamma \varepsilon \{f_{max}, \Sigma f_i\}$. As mentioned in Section 2.3, the following objective functions have frequently been chosen to be minimized:

$$f_{max} \varepsilon \{C_{max}, L_{max}, T_{max}\}$$

or

$$\Sigma f_i \varepsilon \{\Sigma C_i, \Sigma T_i, \Sigma U_i, \Sigma w_i C_i, \Sigma w_i C_i^2, \Sigma w_i T_i, \Sigma w_i U_i\}$$

2.4.4 Examples

$1/r_i/\Sigma w_i C_i$: Minimizing the weighted sum of completion time on a single machine subject to arbitrarily release dates.

$J2/P_{ij}=1/C_{max}$: Minimizing the maximum completion time in a two-machine job shop with unit processing times.

Using the problem classification described in Section 2.4, we can indicate problems under study as follows:

$1/B/\gamma$: In Chapter 5.

$1/r_i/\Sigma w_i C_i$: In Chapter 6.

$1//\Sigma w_i C_i^2$: In Chapter 7.

$F2/prec/C_{max}$: In Chapter 9.

$Pm/prec/C_{max}$: In Chapter 10.

2.5 Computational Complexity

The computation time needed to solve a scheduling problem is obviously of great importance. *P is the set of all decision problems that can be solved by a deterministic algorithm in a time bounded by a polynomial of the input size.* (A decision problem is one whose solution is either "yes" or "no"). *NP is the set of all decision problems that can be solved by a non-deterministic algorithm in a time bound by a polynomial of the input size.* The class NP is very extensive. It is obvious that $P \subseteq NP$. All scheduling problems that will be considered in this thesis can be solved by non-deterministic algorithms and thus are members of NP.

Cook (Cook, 1971) proved that there are hardest problems in NP. Such problems are called NP-complete. *A problem P' is NP-complete if the existence of a polynomial algorithm for P' implies the existence of a polynomial algorithm for any problem in NP* (i.e. P=NP). The location of the borderline separating the "easy" problems (in P) and the hard ones (in NP-complete) has been under wide investigation of many researchers, but turns out that a minor change in the value of an easy problems parameter often transforms this problem into a hard one.

NP-completeness of a problem is generally accepted as strong evidence that the existence of a good algorithm is unlikely and hence enumerative optimization methods such as branch and bound or heuristic methods are to be used.

The optimization version of an existence problem that is NP-complete is called NP-hard.

Finally, a problem is said to be an open problem if a polynomial bounded algorithm for solving this problem has not been found and the problem has not been proved to be hardest in NP.

CHAPTER THREE

METHODS OF APPROACH

3.1 Introduction

The machine scheduling problem is a combinatorial optimization problem. The objective in this kind of problem is to find an optimal schedule among a large but finite number of feasible schedules.

Every schedule is determined by the starting times of all operations, where the starting time of an operation 0_{ij} on machine j, s_{ij} is greater than or equal to the completion times of all other operations of job i that must precede the given operation. *A schedule is called a semi-active schedule if the starting time of no operation can be decreased without changing the processing order on some machine.* Since the class of semi-active schedules Z being in one to one correspondence with feasible sequences, it has finite cardinality of at most $(n!)^m$. It can easily be proved that Z contains at least one optimal schedule with respect to any regular measure (Theorem 2.2, Rinnooy Kan, 1976). For a $Pm/\beta/\gamma$ problem, the number of feasible schedules is bounded from above by n!. This number increases to $(n!)^m$ for the $Jm/\beta/\gamma$ problem. This number, $(n!)^m$, is very large even for small values of n and m. For example, if n = m = 5: $(n!)^m$ = 24,883,200,000.

Some further slight improvement is possible by identifying a subset of Z containing an optimal schedule with respect to any regular measure. This subset is the set, Z_A, of all *active scheduling*, "i.e. *those semi-active schedules in which it is not possible to decrease the starting time of any operation without increasing the starting time of at least one other operation.* The set Z_A of all active schedules is a subset of Z and must contain an optimal schedule with respect to every regular measure" (Rinnooy Kan, 1976).

"In general, although the set of active schedules is usually a proportionately small and proper subset of the set of semiactive schedules, there are still an impossibly large number." (Conway et al., 1967).

Clearly, searching for an optimum schedule among all possible schedules using complete enumeration is not suitable even for problems of small sizes. Thus the complete enumeration method may be rejected immediately.

Most methods of approach (disregarding complete enumeration) try to reduce the size of the set of feasible schedules by eliminating all sequences (or parts of sequences) that are obviously non-optimum: this is because a sequence that is at least as good has been or can be found.

In his book, Rinnooy Kan (Rinnooy Kan, 1976) gives a full review of most known methods of approach to solving machine scheduling problems. These methods are as follows:

1. Complete Enumeration.
2. Combinatorial Analysis.
3. Integer Programming.
4. Branch and Bound.
5. Dynamic Programming.
6. Heuristic Methods.

Combinatorial analysis methods rely on examining the effect a minor change in a particular sequence has on the value of that sequence. This is done by judging the effect of the interchange of two, possibly adjacent, jobs in a sequence.

Several attempts have been made to solve the machine scheduling problem by formulating it as an *integer programming* problem. Five of these attempts can be found in (Rinnooy Kan, 1976). Although this formulation is attractive, there is no effective algorithm to solve the integer programming problem.

Branch and bound methods are among the most popular methods of approach for solving combinatorial programming problems. This is due to their simplicity and their (often) computational efficiency. A branch and bound algorithm is characterized by its branching procedure, lower bounding procedure and its search strategy.

Dynamic programming methods have been used to solve a number of machine scheduling problems, mainly $1/\beta/\gamma$ problems. Here, machine scheduling problems and other combinatorial optimization problems are interpreted as multistage decision problems. At every stage, an equation (based on Bellman's principle of optimality) is used to describe the optimal criterion function (for each subproblem) in terms of the previously obtained ones. A lower bounding procedure can be associated with this approach too. Thus, dynamic programming may be viewed as a tree search method similar to the branch and bound approach, but the main disadvantage is that storage requirements are larger. However, the method has the great advantage that many partial solutions are eliminated without being explored further. For the implementation of dynamic programming methods, we refer to (Held & Karp, 1962; Rinnooy Kan, 1976; Baker & Schrage, 1978A, 1978B; Lawler, 1981).

The final approach to solving scheduling problems is by using *heuristic methods*. Although these methods do not guarantee optimal solutions (unlike the branch and bound and the dynamic programming methods which guarantee the finding of an optimal solution), they dominate all other methods in real life situations.

In the rest of this chapter we shall discuss in more detail two of these methods of approach, namely the branch and bound approach and the heuristic approach. The branch and bound approach is included because of its popularity, wide use and because it will be the main approach we shall be using in the following chapters (except in Chapter 5). The heuristic approach is included because of its importance in real life situations.

3.2 Branch and Bound Approach

As we mentioned before, branch and bound methods are among the most popular and widely used methods to solve combinatorial programming problems. They were developed and first used by Eastman (Eastman, 1959) for the travelling salesman problem and by Land and Doig (Land & Doig, 1960) in the context of mixed integer programming. They were first applied to scheduling problems by (Ignall & Schrage, 1965; Lomnicki, 1965; Brown & Lomnicki, 1966; McMahon & Burton, 1967). "The main reason for their present popularity seems to be the simplicity of the basic principles, combined with easy implementation (see Lenstra & Rinnooy Kan, 1975) and often surprising computational efficiency. However, by their very nature the computational behaviour of these methods remains unpredictable." (Rinnooy Kan, 1976).

A general description of the branch and bound methods will now be given. The set of all possible schedules is divided up into disjoint subsets (this dividing is known as the branching procedure), each of which may contain more than one possible schedule. A lower bound on the value of each solution in a subset is calculated. If the lower bound calculated for a particular subset is greater than or equal to the upper bound (unless mentioned otherwise, this upper bound is initially set to equal a very large number, i.e. a number that is greater than the value of any feasible schedule), this subset is ignored since an optimal schedule must exist in the remaining subsets. These remaining subsets (if any) have to be considered one at a time. One of these subsets is chosen, according to some search strategy, from which to branch. This subset is then divided (as above) into smaller disjoint subsets. As soon as one of these subsets contains one element only, a complete sequence of the jobs should exist. This sequence is evaluated and if its value is less than the current upper bound, this upper bound is then adjusted accordingly.

The procedure is then repeated until all subsets have been considered. The upper bound at the end of this branch and bound procedure is the optimum for the particular problem.

Thus, the branch and bound algorithm is determined by the following.

3.2.1 The bounding procedure

It describes the way in which we calculate the lower bound. The effectiveness of the bound is the most important parameter, since it determines the efficiency of the complete algorithm. We can distinguish between the following methods of obtaining lower bounds.

(a) Relaxation of Constraints

Here, one (or more) of the constraints is relaxed, such that the solution to the resulting problem can be obtained and used as a lower bound for the original problem. For example, a lower bound for the $1/r_i/\Sigma w_i c_i$ problem can be obtained by relaxing the release date constraints (i.e. by setting $r_i=0$ for all i) and solving the resulting problem using Smith's rule: order the jobs in a non-increasing order of w_i/p_i. Lower bounds can also be obtained by setting $p_i=1$ for all i or $w_i=1$ for all i and solving the resulting problem in each case.

Lower bounds may also be obtained by allowing pre-emption (i.e. by relaxing the constraint that each operation once started has to be completed without interruption) and solving the resulting problem. This method is used in Section 6.4 to obtain a lower bound for the $1/r_i/\Sigma C_i$ problem.

Also, for problems with precedence constraints one can obtain lower bounds by relaxing these precedence constraints (or some of them). For example, a lower bound for the $F2/prec/C_{max}$ problem can be obtained by solving the $F2//C_{max}$ problem using Johnson's procedure (Johnson, 1954).

One can also obtain lower bounds by allowing some machine(s) to process more than one job at a time (i.e. relaxing the machine capacity constraint). This method of obtaining lower bounds is used in Chapters 8 and 10.

(b) Lagrangian Relaxation of Constraints

This method of obtaining lower bounds involves, in the first place, the explicit formulation of a problem as an integer (or mixed integer) program. It is based on the observation that many NP-hard problems are in fact "easy" problems made complicated by some side constraints. These complicating constraints are dualized. Two methods exist for finding the values of the multipliers, namely the subgradient optimization and the multiplier adjustment methods. The optimum solution of the Lagrangian problem is a lower bound on the optimal value of the original problem (minimization problems). The multiplier adjustment method of solving the Lagrangian problem is used in Section 6.3 to obtain a lower bound for the $1/r_i/\Sigma w_i C_i$ problem. Further details about these methods can be found in (Geoffrion, 1974; Fisher, 1978; Van Wassenhove, 1979).

(c) Dynamic Programming State Space Relaxation

This method is based on relaxing the state space associated with a given dynamic programming recursion (i.e. reducing the number of states) in such a way that the solution to the relaxed recursion provides a lower bound which could be included in a branch and bound procedure to solve the problem. "This state space relaxation method is analogous to Lagrangian relaxation in integer programming. Constraints in integer programming formulations appear as state variables in dynamic programming recursions and hence constraint relaxation corresponds to state space relaxation." (Christofides, Mingozzi & Toth, 1981). More details about this method can be found in the above reference and its references.

(d) Relaxation of Objective

Here, the objective function is relaxed in order to obtain a lower bound. For example, a lower bound for the $1//\Sigma w_i T_i$ problem can be obtained as follows: (Van Wassenhove, 1979)

$$\text{Minimize} \quad \Sigma w_i \max(C_i - d_i, 0)$$

$$\geq \Sigma w_i' \max(C_i - d_i, 0), \quad w_i \geq w_i'$$

$$\geq \Sigma w_i' (C_i - d_i)$$

$$\geq \Sigma w_i' C_i - \Sigma w_i' d_i$$

Since $w_i' d_i$ is a constant, a lower bound can be obtained by solving the $1//\Sigma w_i' C_i$ problem using Smith's (Smith, 1956) procedure.

3.2.2 The Branching Procedure

It describes the method used to split up a subset of possible schedules. The most usual ones are as follows:

(a) *Sequencing jobs one by one from the beginning (forwards branching).* This is the widely used method, see Chapters 4, 6 and 8.

(b) *Sequencing jobs one by one from the end (backwards branching).* This method proved to be very effective for the tardiness problem (Lenstra, 1977) and the $1/d_i/\Sigma w_i C_i$ problem (Van Wassenhove, 1979).

(c) *At every stage, a job is chosen to be sequenced either at the beginning or at the end according to some heuristic method based on the data of the problem,* see Chapters 8 and 10.

(d) *At every stage, a job is chosen to be sequenced first, last, directly before another job or directly after another job.* See (Kurisu, 1977; Potts, 1980C), also see Chapter 9.

(e) *At every stage, a job is sequenced either before or after another job.* A heuristic can be used to determine this pair of jobs, see (Potts, 1981) and Chapter 7.

3.2.3 The Search Strategy

It indicates a node (each node corresponds to a branch already made) to branch from. One can distinguish between three methods:

(a) *Branching from the node with the smallest lower bound.* This method usually leads to the optimum faster than methods b and c below, but it requires more computer storage to store the required data at every node (Fox et al., 1978).

(b) *Branching from the recently created node.* To save storage space, this method is used for problems given in Chapters 7 and 9.

(c) *Branching from a node with the smallest lower bound amongst the recently created nodes.* This method usually leads to the optimum faster than method b, but it requires more computer storage space. This method is used for problems given in Chapters 6 and 10.

A branch and bound algorithm can be represented using a search tree. This tree usually has up to n nodes (branches) in the first level, each of which will create up to n-1 nodes in the second level, each one of these new nodes, in turn, will create up to n-2 nodes in the third level, ... and one node in the last level of the search tree (except when the branching procedure (2e) is used, in which case two nodes only exist in every level, but the number of levels in the search tree in this case may exceed n (but not n^2) levels).

Here, we have given the basis of a branch and bound algorithm. Besides this, one can include many devices to improve the efficiency of the branch and bound procedure. For example, one might like to include a heuristic method to obtain an upper bound on the optimum. In this thesis, a heuristic is either applied once before applying the branch and bound procedure (as in Chapters 9 and 10) or at every node of the search tree (as in Chapters 6 and 7).

If it is possible to show that an optimum solution will always exist without branching from a particular node, then that node is dominated and can be eliminated. Dominance rules usually specify whether a node can be eliminated before computing its lower bound. When used, dominance rules

are usually applied at every node of the search tree to eliminate as many nodes as possible. The effect of dominance rules has been demonstrated (using test problems) in Chapters 6, 9 and 10.

Although a branch and bound procedure guarantees the finding of an optimum schedule, a suboptimal solution may result if some of the possibly optimum partial schedules have not been explored. This is usually caused by limiting the number of nodes or the time spent on solving the problem to a fixed number or a fixed time respectively. It can also be caused by restricting the search to those schedules within a given percentage of the optimum: in real life situations one might consider accepting a solution within 10% (say) of the optimum, in which case a node at any level of the search tree is eliminated if the lower bound computed at that node is within 10% of the upper bound.

In this thesis, we shall give branch and bound algorithms for solving several scheduling problems (see Chapters 6, 7, 9 and 10).

3.3 Heuristic Methods

It is clear (from the previous section) that the computational requirements to solve a particular scheduling problem using the branch and bound approach might become too time consuming for large problems. In fact, even for relatively small problems, there is no guarantee that a solution can be found quickly.

Heuristic algorithms avoid this drawback, since by using them one can obtain solutions to large problems in a fraction of the time spent on solving them using branch and bound techniques. Also the computation requirements for heuristic algorithms are usually predictable for problems of a given size. The drawback of heuristic methods is that they do not guarantee optimality and in some cases it may even be difficult to judge their effectiveness.

One way to assess the effectiveness of a heuristic is to examine its worst-case behaviour. Suppose that f^* denotes the optimal solution to a given problem, while f^H denotes the corresponding value obtained when the jobs are sequenced using a certain heuristic H. If, whatever the problem data, $f^H \leq \rho f^* + \delta$ for specified constants ρ and δ, where ρ is as small as possible, then ρ is called the worst-case behaviour ratio of heuristic H.

Unfortunately, not enough work has been done on the worst-case behaviour of these heuristics. First results on the worst-case performance of heuristics were due to Graham (Graham, 1966 & 1969). A review of worst-case performance of scheduling heuristics can be found in (Garey, Graham & Johnson, 1978; see also Chapter 5).

We shall start this section by giving two heuristic methods that have attracted attention because of their general applicability. In Section 3.3.1 we shall talk about sampling techniques, the first of these heuristic methods. The second method, priority rules, will be given in Section 3.3.2. Finally, in Section 3.3.3, we shall suggest a third heuristic method, the tree type heuristic. Although this method will be given in Chapter 5 when discussing heuristic methods for obtaining near-optimal solutions for several one machine problems, it is given here because of its general applicability.

3.3.1 Sampling Techniques

This approach is based on the observation by (Heller, 1960; Ashour, 1972) that the number of distinct schedules with C_{max} as their maximum completion time is usually much smaller than the number of distinct semi-active schedules. This indicates that it is possible to study the distribution of the random variable C_{max} over the set of all semi-active schedules. This distribution was proved to be asymptotically normal

(Heller, 1959). If μ and σ denote the unknown parameters of the above distribution, then it is possible using methods of Bayesian analysis to generate random schedules until we have reached a stage where the probability of finding a smaller schedule time in the next experiment is not greater than some given constant α.

This procedure is started by constructing an initial distribution on the parameters μ_0 and σ_0 which are specified according to our initial beliefs on their values. A random schedule is then generated. The value of this schedule is used to update the parameters of the distribution to yield a new distribution with parameters μ_1 and σ_1. This new distribution is used to calculate p, the probability of finding a value C_{max} in the next experiment which is smaller than C^*_{max}, the best schedule time obtained so far. If $p \leq \alpha$, we stop; otherwise the procedure continues in a similar way (i.e. by updating μ_1 and σ_1 to obtain new parameters μ_2 and σ_2, then calculate p, etc.). "We refer to (De Leede and Rinnooy Kan, 1975) for details on the choice of an initial distribution for this particular case. During some actual experiments on a 20 job $P10//C_{max}$ problem with data provided in (Heller, 1960), convergence of the initial distribution to the final one turned out to be relatively independent of the particular prior distribution chosen. A near-optimal schedule was found in a few seconds CPU time after roughly 250 iterations in most cases.

Nevertheless, it appears to us that the Bayesian approach through its dependency on asymptotic results for the distribution of C_{max} is more of academic interest than of great practical use; it seems difficult to generalize this approach to less structured situations." (Rinnooy Kan, 1976).

3.3.2 Priority Rules

Given a set of schedulable operations S, *a priority rule tells us which operation O_{ik} (corresponding to processing job i on machine k) should*

be scheduled next. (Of course, an obvious way to choose an operation is by selecting an operation O_{ik} randomly).

Most priority rules have been developed and can be found in (Gere, 1966; Conway et al., 1967; Day & Hottenstein, 1970; Rinnooy Kan, 1976). Some of these rules will now be given.

1. *Job i has a minimal due date (the earliest due date, EDD, rule).*
2. *Operation O_{ik} has the earliest completion time (ECT rule).*
3. *Operation O_{ik} has the shortest processing time (SPT rule).*
4. *Job i has the smallest (or largest) slack-time (i.e. difference between its due dates and the sum of remaining processing times).*
5. *Job i has minimal (or maximal) sum of remaining processing times (i.e. least (or most) work remaining).*
6. *Job i has minimal (or maximal) number of remaining operations.*
7. *Operation O_{ik} arrived first at machine k (first come, first served (FCFS) or first-in first-out (FIFO) rule).*

Other priority rules can be found in the above given references.

Conway et al. (Conway, Maxwell & Miller, 1967) reported a study by Jeremiah Lalchandani and Schrage which indicates that priority rules work best on non-delay schedules (schedules obtained using priority rule 7) and that the SPT, random scheduling and the least work remaining rules are superior to most other rules on active schedules. Furthermore, Rinnooy Kan (Rinnooy Kan, 1976) reported that rules based on the sum of remaining processing times are slightly better than the SPT rule, which in turn outperforms the random and FIFO rules. He also reported two heuristic rules given by Gere which turn out to be very effective: an *"alternative operation"* rule where job j is preferred to job i (the job originally chosen) if the choice of job i threatens overdue delivery of job j, and the *"look ahead"* rule whereby job i is forced to wait if an urgent job is about to

become available for processing. All priority rules are reported to work almost equally well when bolstered by these two additional rules. Finally, it is reported in Conway et al. (Conway et al., 1967) that Nugent proposed a method based on mixing the random rule with some priority rules. By doing this Nugent was able to vary the amount of randomness that entered into operation selection. Surprisingly, the tests he made consistently lead to better results than those obtained using either of the two methods by itself.

Having obtained a complete ordering of the jobs on a machine, it might be possible to improve this processing order by interchanging pairs of jobs sequenced in adjacent positions. This method is likely to be effective for problems with the same processing order on all machines.

3.3.3 The Tree Type Heuristic

From Section 3.2 we know that although a branch and bound procedure guarantees the finding of an optimum schedule, a suboptimal solution may result if some of the possibly optimum partial schedules have not been explored. This fact has been used to obtain near-optimal solutions for many scheduling problems. Here only some of the candidates, within each level of the tree, are chosen from which to branch. Usually, one candidate only is chosen within each level of the tree. Rarely more than one candidate is chosen within each level of the tree. We can identify the following methods of choosing candidates (Muller-Merbach, 1981).

1. According to some priority rules (see Section 3.3.2).

2. According to the value of the objective function of solution-in-process, i.e. job i is selected to be sequenced after an initial partial sequence π if $f(\pi i) \leq f(\pi j)$ for all jobs j, where f denotes the objective function.

3. According to the value of a lower bound computed at every node (look ahead criterion). Obviously, method 2 above is a special case of this method.

4. According to some second order heuristic, which is applied at every node. Obviously, this second order heuristic has to be computationally much faster than the heuristic under consideration.

It is obvious that if the number of chosen candidates is one, one would select a candidate with the smallest lower bound if method 3 is used and one with the smallest value of the heuristic if method 4 is used.

Although this heuristic method can be applied to all types of machine scheduling problems, it is particularly useful for one machine problems, especially for problems with release dates, due dates, and precedence constraints. Several one machine heuristics can be found in Chapter 5.

PART II

SINGLE MACHINE SCHEDULING

CHAPTER FOUR

SINGLE MACHINE SCHEDULING

4.1 Introduction

In this chapter we shall give a brief review of the principal results in one machine problems. We shall classify these results according to the optimality criterion chosen. Heuristic methods for obtaining near optimum solutions for single machine problems will not be discussed in this chapter; they will be considered in Chapter 5.

Section 4.2 deals with f_{max} criteria. Section 4.3 deals with Σf_i criteria. The $1/\beta/\Sigma w_i C_i$ problem is considered in Section 4.3.1. In Section 4.3.2 we consider the $1/\beta/\Sigma w_i C_i^2$ problem. The $1/\beta/\Sigma w_i T_i$ problem will be dealt with in Section 4.3.3. In Section 4.3.4 we shall be dealing with the $1/\beta/\Sigma w_i U_i$ problem. Criteria with multiple objectives function are considered in Section 4.4.

We conclude this section by giving a theorem which will be applied throughout this chapter.

Theorem 4.1 (Conway et al., 1967)

There exists an optimal schedule with respect to any regular measure for any single machine problem with equal release dates without machine idle time and without job splitting.

4.2 Minimizing Maximum Cost, f_{max}

Lawler (Lawler, 1973) gave an $O(n^2)$ algorithm to solve the 1/prec/ f_{max} problem. His algorithm is considered as the most general result in single machine sequencing. The general step of the algorithm is as follows. Let S denote the set of unscheduled jobs at step h of the algorithm. Let $P(S) = \Sigma_{i \epsilon S}\, p_i$ and let $S' \subseteq S$ denote the set of jobs with no successors in S.

Sequence a job $j \varepsilon S'$ with $f_j(P(S)) \leq f_i(P(S))$ for all $i \varepsilon S'$ in the last available position.

When $f_{max} = L_{max}$, the $1//L_{max}$ problem can be solved in $0(n \log n)$ steps using Jackson's EDD rule, i.e. by ordering the jobs according to non-decreasing due dates (Jackson, 1955). Introducing release dates, the general $1/r_i/L_{max}$ problem is NP-hard (Lenstra et al., 1977). However, when all processing times are equal, we have two solvable cases. The first one arises when $p_i=1$ for all i, in which case the problem can be solved using the extended Jackson's rule: sequence an available job with the smallest due date next (a job i is said to be available to be considered for sequencing in a given position if its release date is less than or equal to the completion time of the job sequenced in the previous position, or if job i has a minimal release date amongst unscheduled jobs). This algorithm is proposed as a heuristic for the problem with general processing times by Schrage (Schrage, 1971). The second case arises when $p_i=p$ for all i for which a more sophisticated algorithm exists (Simons, 1978). With regard to the second case, we have the following. Let $\pi = (\pi(1),\ldots,\pi(n))$ be a schedule obtained using the extended Jackson's rule. If $C_{\pi(h)} \leq d_{\pi(h)}$ for $h=1,\ldots,n$, then the schedule π is optimum; otherwise, let $\pi(i)$ be the first late job in π. If a job $\pi(j)$ where $j<i$ with $d_{\pi(j)} > d_{\pi(i)}$ does not exist, there is no feasible schedule. On the other hand, if there exists such a job $\pi(j)$ there may exist a feasible schedule. Searching for a feasible schedule can be done as follows. Choose j as large as possible and add a constraint that job $\pi(j)$ cannot be scheduled before jobs $\pi(h)$ for $h=j+1,\ldots,i$. This is done by setting $r_{\pi(j)} = \min_{h=j+1,\ldots,i} (r_{\pi(h)})$. The extended Jackson's rule is then applied again subject to the added constraint. The feasibility question is answered in $0(n^3 \log n)$ steps. An improved implementation by Garey et al. (Garey et al., 1981) requires only $0(n \log n)$ steps. (Lawler, Lenstra & Rinnooy Kan, 1981).

Introducing precedence constraints among jobs, the special cases: $1/prec/L_{max}$, $1/prec,r_i,p_i=1/L_{max}$ and $1/prec,r_i,p_i=p/L_{max}$ can still be solved by increasing release dates and decreasing due dates such that if i must be sequenced before j (according to the precedence constraints) then $r_i+p_i \leq r_j$ and $d_i+p_j \leq d_j$ (Lageweg et al., 1976). The algorithms described above are then applied to the problems ignoring the precedence constraints.

Various elegant branch and bound methods exist for solving the $1/prec,r_i/L_{max}$ problem, see (Baker & Su, 1974; McMahon & Florian, 1975; Lageweg et al., 1976; Carlier, 1980).

It is known that any sequence is optimum for the $1//C_{max}$ problem. Introducing release dates, the problem can be solved in 0(n log n) steps by ordering the jobs according to non-decreasing r_i. The procedure can also be used to solve the $1/r_i,prec/C_{max}$ problem after adjusting the release dates to reflect the precedence constraints.

4.3 Minimizing Total Cost, Σf_i

4.3.1 $1/\beta/\Sigma w_iC_i$

The $1//\Sigma w_iC_i$ problem can be solved using Smith's rule: order the jobs according to non-increasing w_i/p_i ratios. This procedure requires 0(n log n) steps. If $w_i=1$ for all i=1,...,n, the procedure reduces to the SPT rule, i.e. ordering the jobs according to non-decreasing processing times.

Adding precedence constraints, represented by a directed graph G, to the problem causes the problem to be NP-hard even if all $p_i=1$ or $w_i=1$ (Lawler, 1978; Lenstra & Rinnooy Kan, 1978).

Branch and bound algorithms for the $1/prec/\Sigma w_iC_i$ problem can be found in (Rinnooy Kan et al., 1975; Potts, 1980C; Potts, 1981). The best

algorithm (to our knowledge) for this problem is the one proposed in (Potts, 1981) which includes results for up to 100 jobs.

The special case in which the precedence graph G is a tree-like graph has been solved by (Horn, 1972); by (Adolphson & Hu, 1973) and by (Sidney, 1975). This procedure requires $O(n \log n)$ steps. If G is a series-parallel graph, the problem can still be solved using an $O(n \log n)$ algorithm derived by Lawler (Lawler, 1978) assuming that the decomposition tree is given. This algorithm is based on forming composite jobs: form a composite job k=ij such that $(i,j)\varepsilon G$, $w_i/p_i < w_j/p_j$ and that j is a direct successor of i. The composite job k can then be treated as one job with $p_k=p_i+p_j$ and $w_k=w_i+w_j$. Starting at the end of the given decomposition tree, the procedure successively forms these composite jobs until an optimum schedule is obtained.

Introducing release dates the $1/r_i/\Sigma C_i$ problem has been shown to be NP-hard (Lenstra et al., 1977). Branch and bound algorithms for this problem have been proposed in (Chandra, 1979) and (Dessouky & Deogun, 1980). For the problem with arbitrary weights, (Rinaldi & Sassano, 1977) have derived several dominance theorems. In Chapter 6, branch and bound algorithms for solving the problem with arbitrary weights are derived. Computational results for up to 50 jobs will also be included.

4.3.2 $1/\beta/\Sigma w_i C_i^2$

Only the $1//\Sigma w_i C_i^2$ problem has been considered by other researchers. The problem is still open. To our knowledge, Townsend (Townsend, 1978) was the first to work on this problem. Among other things, Townsend proposed a bounding procedure based on ordering the jobs in a non-increasing order of w_i/p_i ratios and making an adjustment to allow for the potential improvement that could be obtained by interchanging jobs i and j (for all i and j) if they are not in the right order according to non-increasing weights.

Bagga and Kalra (Bagga & Kalra, 1980) proposed some dominance rules for the problem.

The $1//\Sigma w_i C_i^2$ problem is considered in more detail in Chapter 7 where we propose a branch and bound procedure for solving this problem. Computational results for problems with up to 70 jobs are included. Also, we shall show that the special case where agreeable weights (i.e. $p_i \leq p_j \rightarrow w_i \geq w_j$) are assigned to the jobs can be solved by ordering the jobs according to non-increasing weights.

In Section 7.9 we shall show how to apply our proposed bounding procedure for a more general problem where precedence constraints among jobs exist ($1/prec/\Sigma w_i C_i^2$).

4.3.3 $1/\beta/\Sigma w_i T_i$

The $1//\Sigma T_i$ problem is still open. It is considered to be the most famous open scheduling problem. This problem has the following properties:

First: A schedule π obtained by ordering the jobs in a non-decreasing order of their processing times (SPT-rule) is optimal if $d_{\pi(i)} + p_{\pi(i)} \leq C_{\pi(i+1)}$ for all $i=1,\ldots,n-1$ (Rinnooy Kan, 1976).

Second: A schedule π obtained by ordering the jobs in a non-decreasing order of due dates (EDD-rule) is optimal if $T_i \leq p_i$ for all jobs i sequenced in π (Rinnooy Kan, 1976).

Third: The SPT and EDD schedules are optimal if they are identical (as for example when all p_i or all d_i are equal) (Emmons, 1969).

Lawler (Lawler, 1977) developed a pseudopolynomial algorithm requiring $O(n^4 \Sigma p_i)$ time for solving the $1//\Sigma T_i$ problem.

Introducing precedence constraints yields NP-hardness, even for the $1/prec\ p_i=1/\Sigma T_i$ (Lenstra & Rinnooy Kan, 1978). Also, the $1/r_i/\Sigma T_i$ problem is NP-hard (Lenstra et al., 1979).

If all $p_i=1$, the $1/r_i,p_i=1/\Sigma w_iT_i$ problem can be solved as a linear assignment problem. However, the general $1//\Sigma w_iT_i$ problem has been shown to be NP-hard (Lawler, 1977; Lenstra et al., 1977). This general problem has been subject to extensive study (Emmons, 1969; Srinivasan, 1971; Rinnooy Kan et al., 1975; Fisher, 1976; Baker, 1977; Picard & Queyranne, 1978; Baker & Schrage, 1978A, Van Wassenhove, 1979).

Branch and bound algorithms for this problem were developed and used by many of the above listed researchers. Rinnooy Kan et al. (Rinnooy Kan et al., 1975) used a lower bound obtained by solving assignment problems. A different bound was obtained by Fisher (Fisher, 1976) through Lagrengian relaxation: the constraints that the machine can process one job only at a time was relaxed. Picard et al. (Picard & Queyranne, 1978) put the problem into a time-dependent travelling salesman framework. Relaxing the problem led to a shortest path problem. Finally, Van-Wassenhove (Van Wassenhove, 1979) obtained a bound through Lagrengian relaxation. This time, the relaxed problem is a weighted flow-time problem.

4.3.4 $1/\beta/\Sigma w_iU_i$

The $1//\Sigma U_i$ problem can be solved in $O(n \log n)$ steps by using Moore's algorithm (Moore, 1968): let $\pi = (\pi(1),\ldots,\pi(n))$ be the sequence obtained by ordering the jobs in a non-decreasing order of their due dates. If there exists a job $\pi(i)$ (with i as small as possible) that is completed after its due date, one of the jobs sequenced in the first i positions and with the largest processing time is marked late and is removed from the problem. The procedure ends when all the remaining jobs are completed within their due dates. Sidney (Sidney, 1973) extended this procedure to cover the case where certain specified jobs have to be completed in time. Adding deadlines occurring at or after the jobs' due dates causes the problem to be binary NP-hard (Lawler, 1981A).

If agreeable weights (i.e. $p_i < p_j \rightarrow w_i \geq w_j$) were added, the resulting problem can be solved in $O(n \log n)$ steps using Lawler's algorithm (Lawler, 1976). The problem can also be solved in $O(n \log n)$ steps if agreeable release dates (i.e. $d_i < d_j \rightarrow r_i \leq r_j$) were added (Kise et al., 1978B). However, the $1//\Sigma w_i U_i$ problem has been shown to be NP-hard (Karp, 1972), but can be solved by dynamic programming in $O(n\Sigma p_i)$ steps (Lawler & Moore, 1969). The $1/r_i/\Sigma U_i$ problem has also been shown to be NP-hard (Lenstra, 1977).

Introducing precedence constraints causes the problem to be NP-hard even the $1/prec,p_i=1/\Sigma U_i$ problem (Garey & Johnson, 1976). The special case with chain-like precedence constraints has also been shown to be NP-hard (Lenstra & Rinnooy Kan, 1980).

4.4 Multiple Objectives

Although many real life sequencing problems involve multiple criteria, surprisingly little work has been done on these multiple criteria. The problems we shall consider in this section each involves only two criteria. We can identify three types of multiple criteria problems.

The first of these types of problems involves identifying all sequences that minimize a first objective. One of these sequences which minimizes a second objective is chosen as the optimal sequence for that problem.

The second of these multiple criteria problems involves finding a sequence which minimizes the (weighted) sum of two objectives.

In the last type of these multiple criteria problems we are going to consider both criteria as equally important. This time the problem is to find a sequence that does well on both objectives (if such a sequence exists).

Smith (Smith, 1956) considered a multi-objective problem, where the primary criterion is to complete processing all the jobs before their

deadlines; the secondary criterion is to find a sequence with minimal sum of completion times. This problem is denoted by $1/d_i/\Sigma C_i$. He gave an $O(n \log n)$ algorithm to solve this problem (backward scheduling): Let $P(S) = \Sigma_{i \varepsilon S}\, p_i$, sequence a job j with $d_j \geq P(S)$ and with p_j as large as possible in the last available position. Smith's procedure can also be used to solve the special cases $1/d_i, p_i{=}1/\Sigma w_i C_i$ and the $1/d_i, p_i \leq p_j \rightarrow w_i \geq w_j/\Sigma w_i C_i$ (i.e. the problem with agreeable weights, even local agreeable weights: jobs involved in each step of Smith's procedure have agreeable weights) (Van Wassenhove, 1979).

However, the $1/d_i/\Sigma w_i C_i$ problem has been shown to be NP-hard (Lenstra, 1977). This problem has the following properties (Van Wassenhove, 1979).

1. *Feasibility:* Order the jobs in a non-decreasing order of their due dates (EDD rule). Then a feasible solution to the problem exists if and only if $C_i \leq d_i$ for all jobs i.

2. *Optimality:* Order the jobs in a non-increasing order of w_i/p_i ratios (WSPT rule). Then, if $C_i \leq d_i$ for all jobs i, the sequence is optimal.

Van Wassenhove also proposed a branch and bound procedure to solve this problem. The lower bound is obtained by solving a dual problem obtained through a Lagrangian relaxation of the deadline constraints. Bansal (Bansal, 1980) derived some dominance rules for this problem and presented a branch and bound procedure to solve the problem. This time the lower bound is obtained by ordering the jobs in a non-increasing order of w_i/p_i ratios.

Emmons (Emmons, 1975) considered a slightly different problem. This time, the primary criterion is to minimize the number of late jobs while the secondary criterion is to minimize the sum of completion times. This problem can be looked at as a generalization of the $1//\Sigma U_i$ problem

which can be solved using Moore's algorithm (see Section 4.3.4) and of the $1/d_i/\Sigma C_i$ problem which can be solved using Smith's procedure. Emmons proposed a branch and bound procedure for solving this problem. Each branch corresponds to the assignment of an additional job to the set of late jobs. Once a stage where all the remaining jobs (ordered according to the EDD rule) are completed in time is reached, these jobs are sequenced optimally using Smith's algorithm. Emmons also gave some dominance theorems that eliminate many of the branches at each node of the search tree. The computational experiments he carried out indicated that the solution obtained using Moore's algorithm is usually optimal for problems of small sizes (n=10) and some times optimal for problems of larger sizes, and in any case gives solutions within 1 or 2 percent of the optimum. The results also indicated that the additional computational effort to continue to optimality to be remarkably little, even for problems of large sizes.

A composite objective problem, where the objective is to minimize the sum of weighted tardiness and weighted completion time ($1//\Sigma(h_iC_i + w_iT_i)$), was first suggested by Gelders and Kleindorfer (Gelders & Kleindorfer, 1974). This problem is clearly NP-hard since the simpler version ($1//\Sigma w_iT_i$) is already NP-hard (Lenstra, 1977). Dominance conditions for the problem can be found in (Van-Wassenhove, 1979). Branch and bound procedures for solving the problem, together with some computational experiences can be found in (Gelders & Kleindorfer, 1975; Van-Wassenhove, 1979).

Van-Wassenhove and Gelders (Van-Wassenhove & Gelders, 1980) considered a multi-objective problem where the objective is to minimize two different criteria. These two criteria are the minimization of the total flowtime and the minimization of the maximum tardiness. (Obviously, the first criterion is minimized by ordering the jobs in a non-decreasing order of their processing times (SPT rule), while the second criterion is

minimized by ordering the jobs in a non-decreasing order of their due dates (EDD rule).) The problem is to find a sequence that does well on both objectives (if such a sequence exists). In order to define such a sequence more precisely they used the concept efficiency.

Given a schedule π, let $H(\pi)$ and $T_{max}(\pi)$ be the total flowtime and maximum tardiness respectively of schedule π. A sequence π^* is *efficient* if there exists no sequence π such that:

$$H(\pi) \leq H(\pi^*)$$

and

$$T_{max}(\pi) \leq T_{max}(\pi^*)$$

where at least one relation holds with strict inequality. Similarly, a sequence π is said to dominate another sequence π' if:

$$H(\pi) \leq H(\pi')$$

and

$$T_{max}(\pi) \leq T_{max}(\pi')$$

where at least one relation is a strict inequality.

"Clearly, the decision maker will choose an efficient sequence. Therefore, the researcher's problem is to characterize the set of efficient sequences and to help the decision maker in his search through this set in order to decide upon a final sequence to be implemented." (Van-Wassenhove & Gelders, 1980). An $O(n^2\bar{p} \log n)$ (pseudo-polynomial) algorithm to solve this problem (where $\bar{p}$ is the average processing time), together with some computational results can be found in the above reference.

CHAPTER FIVE

HEURISTICS FOR SINGLE MACHINE PROBLEMS

5.1 Introduction

It is clear that the computational requirements to solve a particular scheduling problem using the branch and bound approach might become too time consuming for large problems. In fact, even for relatively small problems, there is no guarantee that solution can be found quickly. *Heuristic algorithms avoid this drawback since by using them we can obtain solutions to large problems in a fraction of the time spent on solving them using branch and bound techniques. Also, the computation requirements for heuristic algorithms are usually predictable for problems of a given size. The drawback of the heuristic algorithms is that they do not guarantee optimality and in some cases it may even be difficult to judge their effectiveness.*

It is well known that precedence constraints among the jobs can be represented by a *directed acyclic graph* $G=(V,E)$. The vertices of G represent the jobs, and if a directed path from vertex i to vertex j exists, then job i must be processed before job j. The *transitive closure* of the directed graph G is the graph obtained by adding all arcs (i,j) (if they do not already exist) to G whenever there is a directed path from vertex i to vertex j. The *transitive reduction* of G is the graph obtained by deleting all arcs (i,j) from G whenever there is a directed path from vertex i to vertex j which does not include the arc (i,j) itself. Job i is a *predecessor* of job j and job j is a *successor* of job i if the arc (i,j) exists in the transitive closure of G. Job i is a *direct predecessor* of job j and job j is a *direct successor* of job i if the arc (i,j) exists in the transitive reduction of G. Define $B_j = \{i/(i,j)\varepsilon G\}$ and $A_j = \{i/(j,i)\varepsilon G\}$. Let B be the set of jobs with no predecessors (i.e.

$B = \{j/B_j = \phi\}$) and A be the set of jobs with no successors (i.e. $A = \{j/A_j = \phi\}$). Obviously, if $E=\phi$, then $B=A=V$. Finally, we define an unsequenced job i to be *available* for sequencing in the first available position if $i \in B$ and $r_i \leq \max\{T', \min_{j \in B}(r_j)\}$, where T' is the completion time of an initial partial sequence σ'.

In Section 5.2 we shall give five heuristics which appeared in the literature. In Section 5.3 we shall suggest four new heuristics to complete our comprehensive list of one machine heuristics. A heuristic which can be applied to most one machine problems is given in Section 5.4 together with some computational experience, followed by concluding remarks in Section 5.5.

5.2 Heuristics Chosen from the Literature

5.2.1 $1/r_i/L_{max}$

We shall consider an equivalent problem, where each job i (i=1,...,n) has a release date r_i, has a processing time p_i, and has a delivery time q_i. The objective is to find a sequence of jobs that minimizes the time by which all jobs are delivered.

The basis of the first heuristic to be given in this section is to sequence an available job i with q_i as large as possible in the first available position. If there is a choice, the job with the larger processing time is chosen. A formal statement of the algorithm will now be given.

Step 1: Let S' be the set of all unsequenced jobs, let k=0 and find $T' = \min_{j \in S'} \{r_j\}$.

Step 2: Find the set $S'' = \{j/j \in S' \ \& \ r_j \leq T'\}$. Find a job i with $i \in S''$ and $q_i = \max_{j \in S''} \{q_j\}$ (in case of ties choose job i with the largest processing time).

Step 3: Set k=k+1, sequence job i in position k, set $T' = T'+p_i$ and $S' = S'-\{i\}$. If $S'=\emptyset$, then stop; otherwise set $T' = \max\{T', \min_{j\in S'}\{r_j\}\}$ and go to Step 2.

The above heuristic is due to Schrage (Schrage, 1971) and requires $O(n \log n)$ steps.

If $\pi = (\pi(1),\ldots,\pi(n))$ is the sequence obtained using Schrage's algorithm, then the time by which all jobs are delivered T_{Sc} is given by:

$$T_{Sc} = r_{\pi(i)} + \sum_{h=i}^{j} p_{\pi(h)} + q_{\pi(j)}$$

where either $\pi(i)$ is the first job in π or the machine will be idle immediately before it processes job $\pi(i)$, and job $\pi(j)$ is chosen such that $1 \leqslant i \leqslant j \leqslant n$.

Kise et al. (Kise et al., 1978A) have shown that $T_{Sc}/T^* \leqslant 2-3/(SP+1)$, where SP is the sume of processing times of all jobs, T^* is the minimum time by which all jobs are delivered and T_{Sc} denotes the minimum time by which all jobs are delivered when the jobs are sequenced using Schrage's algorithm.

From the construction of π, we have

$$r_{\pi(i)} \leqslant r_{\pi(h)} \qquad \text{for } h=i,\ldots,j$$

If $q_{\pi(j)} \leqslant q_{\pi(h)}$ for $h=i,\ldots,j$ then the sequence π is optimum. Otherwise, we can find a job $\pi(k)$ such that $i \leqslant k < j$ and that $q_{\pi(k)} < q_{\pi(j)}$ but $q_{\pi(h)} \geqslant q_{\pi(j)}$ for $h=k+1,\ldots,j$. Job $\pi(k)$ is called the *interference job* and job $\pi(j)$ is called the *critical job*.

If $p_{\pi(k)}$ is the processing time of the interference job, then it has been shown that $T_{Sc}/T^* < 1 + p_{\pi(k)}/SP$ (Kise & Uno, 1978).

Potts (Potts, 1980B) gave a modified heuristic based on applying Schrage's algorithm successively, each time constraining the interference job to be processed after the critical job in the following sequence. The formal statement of this modified heuristic will now be given.

Step 1: Set $t = 0$ and $T_p = \infty$.

Step 2: Apply Schrage's algorithm. Let $\pi^{(t)} = (\pi^{(t)}(1),\ldots,\pi^{(t)}(n))$ be the resulting sequence and let $T_{Sc}^{(t)}$ denote the minimum time by which all jobs sequenced in $\pi^{(t)}$ are delivered.

Step 3: Set $T_p = \min\{T_p, T_{Sc}^{(t)}\}$.

Step 4: If two jobs $\pi^{(t)}(k)$ and $\pi^{(t)}(j)$ such that $k<j$ (k and j as small as possible) and $q_{\pi^{(t)}(k)} < q_{\pi^{(t)}(j)}$ can be found, then go to Step 5; otherwise go to Step 6.

Step 5: Set $r_{\pi^{(t)}(k)} = r_{\pi^{(t)}(j)}$ and $t=t+1$. If $t=n$, go to Step 6; otherwise go to Step 2.

Step 6: Stop with the sequence obtained has T_p as the minimum time by which all jobs are delivered.

The worst case performance of this modified heuristic is 3/2, i.e. $T_p/T^* < 3/2$, and it requires $O(n^2 \log n)$ steps (Potts, 1980B).

If precedence constraints among jobs (represented by a directed graph G in which jobs are renumbered such that an arc (i,j) in G implies that $i<j$) were introduced to the problem, then Schrage's algorithm can still be used as a heuristic for the resulting problem after making the following adjustment

$$r_i = \max\{r_i, \max\{r_j+p_j/(j,i)\varepsilon G\}\}$$

and

$$q_i = \max\{q_i, \max\{q_j+p_j/(i,j)\varepsilon G\}\}$$

for all jobs $i=1,\ldots,n$ and in that order. This heuristic requires $O(n^3)$ steps, which are needed to compute the transitive closure of the precedence graph G.

If $r_i=0$ for $i=1,\ldots,n$, then Schrage's algorithm is optimum even for the constrained problem in which case q_i, $i=2,\ldots,n$ are adjusted as above (Baker, 1974).

5.2.2 $1/r_i/\Sigma C_i$

The following heuristic is based on sequencing a job i with the earliest completion time in the first available position. A formal

statement of the algorithm will now be given.

Step 1: Let S' be the set of all unsequenced jobs and let $T'=0$.

Step 2: Find a job $i \epsilon S'$ with $\max(T', r_i)+p_i$ as small as possible-

Step 3: Sequence job i in the first available position. Set $T' = \max(T', r_i)+p_i$ and $S'=S'-\{i\}$. If $S'=\phi$, stop; otherwise go to Step 2.

This heuristic is due to Van Wassenhove (Van Wassenhove, 1979) and requires $0(n \log n)$ steps.

A sequence $\pi = \{\pi(1),\ldots,\pi(n)\}$ obtained using the above heuristic is optimum if $r_{\pi(1)} \leqslant r_{\pi(j)}$, for j=2,...,n and that for each job $\pi(i)$ with $r_{\pi(i)} \geqslant C_{\pi(i-1)}$, we have:

$$r_{\pi(i)} \leqslant r_{\pi(j)}, \quad \text{for all } j=i,i+1,\ldots,n$$

5.2.3 $1/d_i/\Sigma w_i C_i$

The following heuristic is due to Smith (Smith, 1956). The basis of this heuristic is to find an unsequenced job j with $d_j \geqslant \sum_{i \epsilon S'} p_i$ (S' is the set of unsequenced jobs) and w_j/p_j as small as possible. This job is then sequenced in the last available position. The formal statement of this heuristic will now be given.

Step 1: Let S' be the set of all unsequenced jobs and let $T' = \sum_{i \epsilon S'} p_i$.

Step 2: Set $S''=\{i/d_i \geqslant T'\}$. Find a job $j \epsilon S''$ with w_j/p_j as small as possible. Break ties by choosing job j with the smallest processing time.

Step 3: Sequence job j in the last available position and set $S'=S'-\{j\}$. If $S'=\phi$, stop; otherwise go to Step 1.

This heuristic requires $0(n \log n)$ steps.

If precedence constraints among jobs exist, the above procedure can still be used as a heuristic for the resulting problem, except that in this case S'' of Step 2 becomes $S'' = \{i/i \epsilon A \text{ and } d_i \geqslant T'\}$. The heuristic now requires $0(n^3)$ steps.

If all $w_i=1$ for $i=1,\ldots,n$, then the sequence obtained using Smith's heuristic is optimum (Smith, 1956).

5.2.4 $1/prec/\Sigma w_i C_i$

The first heuristic to be given in this section is based on using the following results which have been proved by Morton and Dharan (Morton & Dharan, 1978).

Theorem 5.1

If job i has no predecessors and $w_i/p_i \geq w_j/p_j$ for all jobs j, then there exists an optimum sequence in which job i is sequenced first.

Corollary 5.1

If job i has no successors and $w_i/p_i \leq w_j/p_j$ for all jobs j, then there exists an optimum sequence in which job i is sequenced last.

Theorem 5.2

If job i has at least one predecessor and $w_i/p_i \geq w_j/p_j$ for all jobs j, then there exists an optimum sequence in which job i is sequenced immediately after one of its direct predecessors.

Corollary 5.2

If job i has at least one successor and $w_i/p_i \leq w_j/p_j$ for all jobs j, then there exists an optimum sequence in which job i is sequenced immediately before one of its direct successors.

Every decision made (using the first heuristic) will sequence a job either first (Theorem 5.1), last (Corollary 5.1), immediately after one of its direct predecessors (Theorem 5.2), or immediately before one of its direct successors (Corollary 5.2).

A formal statement of the algorithm will now be given.

Step 1: Let S' be the set of all unsequenced jobs.

Step 2: Find jobs $i \in S'$ and $i' \in S'$ such that $w_i/p_i \geq w_j/p_j$ and $w_{i'}/p_{i'} \leq w_j/p_j$ for all unsequenced jobs j. If i or i' is not uniquely

defined, then an arbitrary choice is made. Let k and k' denote the number of direct predecessors and direct successors of i and i' respectively.

Step 3: If $k=0$, then sequence job i first, and set $S'=S'-\{i\}$. If $S'=\phi$, stop; otherwise go to Step 1.

Step 4: If $k'=0$, then sequence job i' last and set $S'=S'-\{i'\}$. If $S'=\phi$, stop; otherwise go to Step 1.

Step 5: If $0 < k \leq k'$, sequence job i immediately after job j (i.e. form a composite job ji), one of its direct predecessors, with w_j/p_j as small as possible and go to Step 1. Otherwise sequence job i' immediately before job j' (i.e. form a composite job $i'j'$), one of its direct successors, with $w_{j'}/p_{j'}$ as large as possible and go to Step 1.

If two jobs i and j are to be sequenced immediately after each other (Step 5), then these two jobs are replaced by a single (composite) job ij with processing time p_i+p_j and weight w_i+w_j. The precedence graph G is also updated as follows.

(a) The two vertices i and j are replaced by a new vertex $k=ij$.

(b) For each arc (h,i) or (h,j) in G, where $h \neq i$, an arc (h,k) is added.

(c) For each arc (i,h) or (j,h) in G, where $h \neq j$, an arc (k,h) is added.

This heuristic is due to Morton and Dharan (Morton & Dharan, 1978). It is clear that applying the above procedure requires obtaining the transitive reduction of the precedence graph G which in turn requires $O(n^3)$ steps to compute.

Another heuristic for the same problem can be found in the above reference. This heuristic is based on Sidney's decomposition principle.

The basis of this heuristic is to form simple sets D_j, where D_j consists of job j and all its predecessors. One of these sets D_j is chosen such that $\sum_{i \in D_j} w_i / \sum_{i \in D_j} p_i$ as large as possible. If D_j consists of job j only, this job is sequenced first; otherwise job j is removed from D_j, i.e. job j cannot be sequenced in the first available position. The set D_j is then used to form new simple sets as above. The procedure is repeated until all jobs are sequenced. The formal statement of this algorithm will now be given.

Step 1: Let S' be the set of all unsequenced jobs. Let $D=S'$.

Step 2: For each job $i \in D$ with no successors in D, find a set $D_i = \{i\} \cup \{j / j \in D \text{ and } (j,i) \in G\}$.

Step 3: Find a set $D_j \subseteq D$ (one of the sets found in Step 2) with $\sum_{i \in D_j} w_i / \sum_{i \in D_j} p_i$ as large as possible.

Step 4: If D_j consists of job j only, sequence this job first and set $S'=S'-\{j\}$. If $S'=\phi$, stop; otherwise go to Step 1. If D_j consists of more than one job, set $D=D_j-\{j\}$ and go to Step 2.

This heuristic requires $O(n^3)$ steps. Computational experience reported in (Morton & Dharan, 1978) showed the Sidney type heuristic to be computationally almost as good as the first heuristic.

5.2.5 $1//\Sigma T_i$

The following heuristic is known as the Wilkenson-Irwin heuristic (Baker, 1974). It is based on the observation that it is preferable to have job i with the smallest d_i to be sequenced in the first available position except when a job j exists such that $T' + \max\{p_i, p_j\} > \max\{d_i, d_j\}$ (where T' is the sum of processing times of all jobs sequenced in an initial partial sequence π), in which case a job with smallest processing

time is sequenced in the first available position. The algorithm involves two sequences; a sequence π of scheduled jobs and a sequence S' of unscheduled jobs. Jobs sequenced in π are subject to possible revision. The sequence S' contains the remaining jobs in EDD (earliest due dates) order.

At each stage, the algorithm applies the above result to jobs i and j, where $d_i \leq d_j$ (i and j are initialized to be the first two jobs in S'). If $T' + \max\{p_i, p_j\} \leq d_j$ or $p_i \leq p_j$, then job i is added to the end of π. If these conditions fail, the decision rule is applied to jobs k and j, where k is the last job in π. If $T' - p_k + \max\{p_k, p_j\} \leq \max\{d_k, d_j\}$ or if $p_k \leq p_j$ (or if no job exists in π), then job j is added to the end of π. However, if this decision rule fails also, a jump condition results in which case job k is removed from π and replaced in S' in EDD order. The decision rule is then applied to job j and the last job in π. The jump condition occurs infrequently, but it may be applied several times in succession in order to sequence job j. A formal statement of the algorithm will now be given.

Step 1: Let S' be the set of all unsequenced jobs. Let T'=0 and $\{\pi\}=\phi$.

Step 2: Order jobs in S' in a non-decreasing order of d_i.

Step 3: Let i and j be the first two jobs in S'.

Step 4: If $T' + \max\{p_i, p_j\} \leq \max\{d_i, d_j\}$ or $p_i \leq p_j$, then sequence job i next, set $T' = T' + p_i$, remove job i from S' and set i=j. If job i is the only job in S', sequence job i last and stop. If there exist more than one job in S', let j be the second job in S' and repeat Step 4. If, on the other hand, $T' + \max\{p_i, p_j\} > \max\{d_i, d_j\}$ and $p_i > p_j$, then set i=j, let k be the last job in π and proceed to Step 5.

Step 5: If $T' - p_k + \max\{p_k, p_i\} \leq \max\{d_k, d_i\}$ or if $p_k \leq p_i$, then sequence job i next, set $T' = T' + p_i$, remove job i from S' and go to

Step 3. If, on the other hand, $T' - p_k + \max\{p_k, p_i\} > \max\{d_k, d_i\}$ and $p_k > p_i$ then a jump condition results. Go to Step 6.

Step 6: (Jump condition) Remove job k from π, return it to S' in EDD order and set $T' = T' - p_k$. If there exist at least one job in π, let k be the last job in π and go to Step 5. If no job exists in π, sequence job i first in π, set $T' = T' + p_i$ and go to Step 3.

The above heuristic is not polynomially bounded.

If we define the tardiness interval for job i as follows:

$$t_i \text{ is empty if } C_i \leq d_i$$

$$t_i = [d_i, C_i] \text{ if } C_i > d_i$$

Then a sequence obtained using the above algorithm is optimum if there is no time t for which $t \varepsilon t_i$ and $t \varepsilon t_j$ for the tardiness intervals of any pair of jobs i and j (Theorem 2.8, Baker, 1974), i.e. the tardiness intervals are mutually disjoint.

5.3 New Heuristics

In the previous section we gave heuristic procedures for the $1/r_i/L_{max}$, for the $1/r_i/\Sigma C_i$, for the $1/d_i/\Sigma w_i C_i$, for the $1/prec/\Sigma w_i C_i$ and for the $1//\Sigma T_i$ problems. In this section, we shall suggest new procedures to complete our comprehensive list of heuristics. Each of these procedures is written for a problem in its general form, i.e. with release dates and precedence constraints, and hence can be used if r_i or prec is dropped.

The heuristicsproposed in this section are for the $1/r_i,prec/\Sigma w_i C_i$, $1/r_i,prec/\Sigma w_i C_i^2$, $1/r_i,prec/\Sigma w_i T_i$ and the $1/r_i,prec/\Sigma w_i U_i$ problems. The case where each job has both a due date and a deadline is not considered since this case has seldom been considered by researchers. Another case

where each job has a release date and a deadline is not considered either since the existence of a feasible sequence, in this case, is NP-hard (Lenstra, Rinnooy Kan & Brucker, 1977).

5.3.1 $1/r_i,prec/\Sigma w_i C_i$

The heuristic described here is a generalisation of the heuristic proposed for use in calculating a lower bound in the branch and bound procedures proposed in Chapter 6 for solving the $1/r_i/\Sigma w_i C_i$ problems.

Every stage will sequence an available job in the first unfilled position. If there is a choice, one with the largest w_i/p_i is chosen, i.e. according to Smith's rule. A formal statement of the method is given below.

Step 1: Let S' be the set of all unsequenced jobs. Find the set B = $\{j/\text{for each } i \in S', (i,j) \notin G\}$. Also find $T' = \min_{j \in B}\{r_j\}$.

Step 2: Find the set $S'' = \{j/j \in B \text{ and } r_j \leq T'\}$ and find a job i with $i \in S''$ and with $w_i/p_i = \max_{j \in S''}\{w_j/p_j\}$.

Step 3: Sequence job i in the first available position. Set $T' = T'+p_i$ and $S' = S'-\{i\}$. If $S'=\phi$, stop; otherwise find the set of jobs B (as in Step 1), set $T' = \max\{T', \min_{j \in B}\{r_j\}\}$ and go to Step 2.

The above procedure requires $O(n^3)$ steps. If all jobs are independent (i.e. no precedence constraints), then the procedure requires $O(n \log n)$ steps.

Now we shall show that an upper bound on the worst-case performance of this heuristic does not exist. Consider the following two jobs example with $r_1=0$, $p_1=h-2$, $w_1=1$, $r_2=1$, $p_2=1$ and $w_2=h$, for $h \geq 4$. We shall assume that the two jobs are independent.

The above heuristic H sequences job 1 before job 2 yielding sum of weighted completion time $SWCT^H = h^2-2$. However, in the optimum sequence, job 2 is sequenced before job 1 yielding $SWCT^* = 3h$. Thus $SWCT^H/SWCT^* = (h^2-2)/3h$ which can be arbitrarily large.

5.3.2 $1/r_i,prec/\Sigma w_i C_i^2$

As a heuristic for this problem we suggest to use the same heuristic proposed in the previous section for the $1/r_i,prec/\Sigma w_i C_i$ problem.

However, for the $1//\Sigma w_i C_i^2$ problem, we have the following. Let $\pi = (\pi(1),\ldots,\pi(n))$ be the sequence obtained using the heuristic procedure of the previous section, i.e. by ordering the jobs in a non-increasing order of w_i/p_i ratios. Then in order to improve the sequence π, two jobs $\pi(i)$ and $\pi(j)$ are temporarily sequenced in positions j and i ($i<j$) respectively, i.e. job i and job j are temporarily interchanged. If an improvement is made then the two jobs $\pi(i)$ and $\pi(j)$ are left in their new positions. The procedure is then repeated from the beginning (i.e. $i=1$ and $j=2$). If, on the other hand, no improvement can be made, the two jobs $\pi(i)$ and $\pi(j)$ are replaced in their original positions (i.e. position i and position j) and other possibilities are considered in a similar way. The algorithm terminates when all possibilities ($i=1,\ldots,n-1$ and $j=i+1,\ldots,n$) are considered without making any improvement.

This procedure requires $0(n^2)$ steps if the sequence π is optimum. Otherwise the heuristic is not polynomially bounded.

5.3.3 $1/r_i,prec/\Sigma w_i T_i$

As the heuristic proposed in Section 5.3.1, this heuristic has the property that the machine will never be kept unnecessarily idle. If there is a choice of jobs for the first unfilled position, then one of these jobs is chosen as follows. If an available job i is late and with w_i/p_i as large as possible, then this job is sequenced first. Otherwise, an available job i with d_i as small as possible is temporarily sequenced in the first unfilled position. If sequencing job i first will make an available job j (available at time T') with w_j/p_j as large as possible late and with the cost associated with the order ji is less than the cost associated with

the order ij, then job i is removed from its temporary position and job j is sequenced in that position. Otherwise, job i is sequenced permanently in its temporary position. A formal statement of the algorithm will now be given.

Step 1: Let S' be the set of all unsequenced jobs. Find the set $B = \{j / \text{for each job } i \varepsilon S', (i,j) \notin G\}$. Also, find $T' = \min_{j \varepsilon B}\{r_j\}$.

Step 2: Find the set $S'' = \{j / j \varepsilon B \text{ and } r_j \leq T'\}$.

Step 3: If there exists one job only in S'', sequence this job first, set $T' = T'+p_i$ and go to Step 7. Otherwise, proceed to Step 4.

Step 4: Let $T'' = T' + \sum_{i \varepsilon S''} p_i$. If no job j with $j \varepsilon S''$ and with $d_j \geq T''$, go to Step 5. Otherwise, a job j with $j \varepsilon S''$, $d_j \geq T''$ and w_j/p_j as small as possible is removed from the set S'' (i.e. job j will not be sequenced first). Go to Step 3.

Step 5: If there exists a job i with $i \varepsilon S''$ and with $T' + p_i > d_i$ and w_i/p_i as large as possible (if there is a choice), then sequence job i first, set $T' = T'+p_i$ and go to Step 7. Otherwise, proceed to Step 6.

Step 6: Find a job i with $i \varepsilon S''$ and d_i as small as small as possible (Break ties by the SPT rule). Find the set $S''_L = \{k / k \varepsilon S'' \text{ and } T' + p_i + p_k > d_k\}$. If $S''_L \neq \phi$, find a job j $(j \neq i)$ with $j \varepsilon S''_L$ and with w_j/p_j as large as possible. If $S''_L = \phi$ or if $(T' + p_i + p_j - d_j)\, w_j \leq (T' + p_j + p_i - d_i)\, w_i$, then sequence job i first, set $T' = T'+p_i$ and go to Step 7. Otherwise, sequence job j first, set $T' = T'+p_j$ and go to Step 7.

Step 7: Remove the newly sequenced job from S'. If $S' = \phi$, stop; otherwise find the set B (as in Step 1), find $T' = \max\{T', \min_{j \varepsilon B}\{r_j\}\}$ and go to Step 2.

This heuristic requires $O(n^3)$ steps.

If only one job exists in S'', then Step 3 of the above procedure will sequence this job first.

Step 4 will remove a job $j \in S''$, which will be completed in time if sequenced after all other jobs in S'', from being a candidate for the first available position.

Steps 3 and 4 are repeated until one job only exists in S'', in which case this job is sequenced first, or until no job can be removed from S'', in which case Step 5 is executed.

Step 5 will sequence a late job $i \in S''$ with w_i/p_i as large as possible, if such a job exists, in the first available position. If each of the jobs in S'' can be completed in time, if sequenced first, then Step 6 is executed.

Step 6 will first find a job $i \in S''$ with d_i as small as possible and a job $j \in S''$ $(j \neq i)$ with $T' + p_i + p_j > d_j$ and w_j/p_j as large as possible, if such a job j exists. If job j does not exist or if the cost associated with the order ij is less than or equal to the cost associated with the order ji, then job i is sequenced first; otherwise job j is sequenced first.

5.3.4 $1/r_i,prec/\Sigma w_iU_i$

We start each stage of the algorithm by considering all jobs with no successors. The earliest completion time of each of these jobs is then computed. Any job with an earliest completion time which is larger than its due date is sequenced in the last available position. This part of the algorithm is repeated until no job can be sequenced last.

We then consider the set of available jobs; one of these jobs is either sequenced in the first available position, has its release date increased, or is sequenced immediately before one of its direct successors. Sequencing a job i immediately before one of its direct successors j means forming a composite job k=ij where $r_k=r_i$ (we assume that the release dates have been adjusted according to the precedence constraints, as given in

Section 5.2.1) $p_k=p_i+p_j$ and $w_k=w_j$. The precedence graph G is then updated as given in Section 5.2.4. Each composite job k=ij we form will then be treated as a single job.

The procedure is repeated until all jobs have been assigned positions in the sequence. A formal statement of the algorithm will now be given.

Step 1: Let S' be the set of all unsequenced jobs. Set T'=0.

Step 2: Find the set of jobs $A = \{i/\text{for } j\varepsilon S', (i,j)\notin G\}$. For each job i with $i\varepsilon A$, compute C_i, the earliest completion time of job i. If $C_i > d_i$, then sequence job i in the last available position, remove job i from S' and go to Step 8. Otherwise, proceed to Step 3.

Step 3: Find the set of jobs $B = \{j/\text{for } i\varepsilon S', (i,j)\notin G\}$. Also, find $T' = \max\{T', \min_{j\varepsilon B}\{r_j\}\}$ and $B' = \{j/j\varepsilon B \text{ and } r_j \leq T'\}$.

Step 4: Find the set of jobs $B'_L = \{j/j\varepsilon B' \text{ and } T' + p_j > d_j\}$. If $B'_L=\emptyset$, go to Step 5. Otherwise, find a job $i\varepsilon B'_L$ with p_i as small as possible and proceed to Step 4.1.

Step 4.1: If $B'_L=B'$, sequence job i next, set $T'=T'+p_i$, remove job i from S' and go to Step 8. Otherwise proceed to Step 4.2.

Step 4.2: Let A_i be the set of direct successors j of i. Find the set of jobs $A'_i = \{j/j\varepsilon A_i \text{ and } T' + p_i \geq r_j\}$. Let k_i denote the number of jobs in A'_i.

Step 4.2.1: If $k_i=0$, set $r_i = \max\{r_i, \min_{j\varepsilon A_i}\{r_j\}-p_i\}$ and go to Step 3.

Step 4.2.2: If $k_i=1$, form a composite job ij, $j\varepsilon A'_i$ and go to Step 2.

Step 4.2.3: If $k_i>1$, find the set $A''_i = \{j/j\varepsilon A'_i \text{ and } T'+p_i+p_j\leq d_j\}$. Let n_i denote the number of jobs in A''_i.

Step 4.2.3.1: If $n_i=0$, form a composite job ij, where $j\varepsilon A'_i$ with p_j as small as possible and go to Step 2.

Step 4.2.3.2: If $n_i=1$, form a composite job ij, where $j \varepsilon A_i''$ and go to Step 2.

Step 4.2.3.3: If $n_i>1$, find jobs j and j' where $j,j' \varepsilon A_i''$ with d_j as small as possible and $w_{j'}/p_{j'}$ as large as possible. If $T'+p_i+p_j+p_{j'} \leq d_{j'}$ form a composite job ij; otherwise form a composite job ij'. Go to Step 2.

Step 5: If only one job i exists in B', sequence this job first, set $T'=T'+p_i$, remove job i from S' and go to Step 8. Otherwise proceed to Step 6.

Step 6: Let $T''=T'+\sum_{i \varepsilon B'} p_i$. If there exists a job $j \varepsilon B'$ with $d_j \geq T''$ (and w_j/p_j as small as possible), then remove job j from B' (i.e. job j cannot be sequenced first) and go to Step 5. Otherwise proceed to Step 7.

Step 7: Find a job $i \varepsilon B'$ with d_i as small as possible. Find the set of jobs $B'' = \{j/j \varepsilon B' \text{ and } T'+p_i+p_j>d_j\}$. If $B'' \neq \emptyset$, find a job $j \varepsilon B''$ with w_j/p_j as large as possible. If $B''=\emptyset$ or $w_i/p_i \geq w_j/p_j$, sequence job i first, set $T'=T'+p_i$, remove job i from S' and go to Step 8. Otherwise, sequence job j first, set $T'=T'+p_j$, remove job j from S' and go to Step 8.

Step 8: If $S'=\emptyset$, stop; otherwise go to Step 2.

This procedure requires $0(n^3)$ steps.

Step 2 of the algorithm will compute the earliest completion time (based on r_i and prec) of each job i with no successors. Each time a late job is found, this job is removed from the set of unsequenced jobs S' and sequenced in the last available position. If all jobs have been sequenced, the procedure ends; otherwise Step 2 is repeated (Step 8). Steps 2 and 8 are repeated until all jobs have been sequenced or until a stage where none of the jobs with no successors is late, in which case Step 3 is executed.

Step 3 will find B', the set of available jobs which is needed in the following steps of the algorithm.

If none of the available jobs is late (Step 4), then Step 5 is executed. Otherwise, an available job i, where job i is late and with p_i as small as possible is found and we have the following. If all available jobs are late, then job i is sequenced first (Step 4.1). If none of the direct successors of job i is available at time $T'+p_i$ (i.e. assuming job i is sequenced next), then job i is delayed possibly by increasing its release date (Step 4.2.1). If only one of the direct successors of job i (j say) is available at time $T'+p_i$, then a composite job ij is formed (Step 4.2.2). If more than one of the direct successors of job i are available at time $T'+p_i$, we have the following. If none of these jobs can be completed before its deadline when sequenced at time $T'+p_i$, then a job j (one of these jobs) with p_j as small as possible is found and a composite job ij is formed (Step 4.2.3.1). If only one of these jobs, say j, (i.e. direct successors of job i which are available at time $T'+p_i$) can be completed before its deadline when sequenced at time $T'+p_i$, then a composite job ij is formed (Step 4.2.3.2). Finally, if more than one job can be completed before their deadlines when each of them in turn is sequenced at time $T'+p_i$, then two of these jobs j and j' are chosen (we may have j=j') such that d_j is as small as possible and $w_{j'}/p_{j'}$ is as large as possible. If job j' can be completed before its deadline when sequenced at time $T'+p_i+p_j$ (i.e. assuming jobs i and j have been sequenced), then a composite job ij is formed; otherwise a composite ij' is formed.

Steps 5, 6 and 7 will deal with the case where each of the available jobs $j \epsilon B'$ (found in Step 3) can be completed before its deadline if it is sequenced in the first available position. The only access to these steps is from Step 4.

Step 6 reduces, if possible, the number of available jobs by temporarily removing any job which, when sequenced after all other available jobs, can be completed before its deadline. Obviously, if only one job is left, then this job is sequenced first (Step 5).

Finally, Step 7 will find an available job i with d_i as small as possible and a job j which is going to be late when sequenced at time $T'+p_i$ (i.e. assuming job i is sequenced first) and with w_j/p_j as large as possible, if such a job j can be found. If no such job j exists or if $w_i/p_i \geq w_j/p_j$, then job i is sequenced first; otherwise, job j is sequenced first.

We point out that if prec is dropped, then Steps 4.1, 4.2, 4.2.1, 4.2.2, 4.2.3, 4.2.3.1, 4.2.3.2 and 4.2.3.3 of the algorithm are not executed.

5.4 The Tree Type Heuristic

5.4.1 The Algorithm

From Section 3.2 we know that although a branch and bound procedure guarantees the finding of an optimum schedule, a suboptimal solution may result if some of the possibly optimum partial schedules have not been explored. This fact has been used to obtain near-optimum solutions for many scheduling problems. Here, suitable dominance rules can be used to reduce the number of candidates within each level of the tree. Then only some of the remaining candidates (within each level of the tree) are chosen from which to branch. Usually, one candidate only is chosen within each level of the tree. Rarely, more than one candidate is chosen within each level of the tree. Methods of choosing candidates can be found in (Muller-Merbach, 1981; Section 3.3.3). Here, we suggest two methods to choose one candidate only to branch from within each level of the tree.

(a) According to the value of a lower bound computed at every node (look ahead criterion). We shall refer to this case by H_L.

(b) According to some second order heuristic H, which is applied at every node. Obviously, this second order heuristic H has to be fast computationally. Several one machine heuristics can be found in Sections 5.2 and 5.3. We shall refer to this case by H_H.

It is obvious that since the number of chosen candidates is one, one would select a candidate with the smallest lower bound if method 'a' is used and one with the smallest value of the heuristic if method 'b' is used.

It is clear that a branch and bound procedure will lead to the same solution as method 'a' (using the same lower bound) within the same number of nodes and thus it may be useful to include method 'b' in the branch and bound procedure, since this may lead to a different and may even be a better solution.

The performance of these two methods of choosing one candidate within each level of the tree to obtain a near optimum solution for the $1/r_i/\Sigma w_i C_i$ problem was assessed using test problems. The branching procedure (Forwards Branching, FB), the lower bounding procedure (the improved lower bound LB') and the second order heuristic used are those proposed in Chapter 6. The performance of the second order heuristic H (see also Section 5.3.1) on its own was also tested using the same problems.

This tree type heuristic requires $O(n^3 \log n)$ if the second order heuristic H is used and $O(n^4 \log n)$ if the improved lower bound LB' of Chapter 6 is used.

5.4.2 Computational Experience

5.4.2.1 Test Problems

Every problem consists of n jobs where n=20, n=30, n=40 or n=50. Three integers were generated for every job i, namely p_i, w_i and r_i.

Processing times p_i and weights w_i were generated randomly from uniform distributions [1,100] and [1,10] respectively. Release dates for every problem were generated from the uniform distribution [0,50.5nR], where R controls the range of the distribution. The value 50.5n measures the expected total processing time. For each selected value of n, five problems were generated for each of the R values 0.2, 0.4, 0.6, 0.8, 1.0, 1.25, 1.5, 1.75, 2.0 and 3.0 producing fifty problems for each value of n. This method of data generation follows that given in Chapter 6.

5.4.2.2 Computational Results

Computational results for the second order heuristic H are given in Table 5.1, while the results for the two tree type heuristics H_H and H_L are given in Table 5.2. The branch and bound procedures of Chapter 6 were used to solve the tested problems. As we shall point out in Chapter 6, whenever a problem was not solved within the time limit of 60 seconds, computation was abandoned for that problem. (In all, 5 and 21 problems were left unsolved when n=40 and 50 respectively). Thus, in some cases the figures given in Tables 5.1 and 5.2 are lower bounds on the average and maximum deviations and upper bounds on the number of problems with values within a given percentage of the optimum.

Results for heuristic H are given in Table 5.1. The first column of Table 5.1 shows the average deviations (%) of this heuristic. This average takes its maximum value 1.08% when n=20. This value decreases as n increases and takes its minimum value 0.34% when n=50. This is due to the fact that when n increases, the effect each individual job has on the value of the heuristic decreases.

Column two shows that when n=20, six problems have optimum sequences. As expected, this figure decreases as n increases and takes its minimum value of 0 when n=50.

Table 5.1: Results for Heuristic H

n	Average Deviation %	Number of Cases Within % of Optimum						Maximum Deviation %
		0.0	1.0	2.0	3.0	4.0	5.0	
20	1.08	6	36	42	45	47	48	8.62
30	0.98	3	34	40	46	49	50	4.38
40*	0.64	2	43	45	48	50	50	3.93
50*	0.34	0	47	50	50	50	50	1.52

*Lower bounds on the average and maximum deviations and upper bounds on the number of cases within % of optimum because of unsolved problems.

Table 5.2: Results for Heuristics H_H and H_L

n	Heur-istic	Average Deviation %	Number of Cases Within % of Optimum						Maximum Deviation %	N_H
			0.0	1.0	2.0	3.0	4.0	5.0		
20	H_H	0.37	27	45	48	49	49	49	8.28	3
	H_L	0.03	41	50	50	50	50	50	0.64	18
	Best	0.03	42	50	50	50	50	50	0.64	--
30	H_H	0.24	22	47	49	50	50	50	2.71	6
	H_L	0.05	29	50	50	50	50	50	0.56	26
	Best	0.04	35	50	50	50	50	50	0.56	--
40*	H_H	0.19	17	49	50	50	50	50	1.57	11
	H_L	0.07	23	50	50	50	50	50	0.61	25
	Best	0.05	28	50	50	50	50	50	0.36	--
50*	H_H	0.13	11	50	50	50	50	50	0.70	11
	H_L	0.04	18	50	50	50	50	50	0.20	31
	Best	0.03	22	50	50	50	50	50	0.20	--

*As above. N_H = Number of cases a heuristic is better than the other.

The next five columns of the same table show all problems of sizes 30, 40 and 50 have solutions within 5% of optimum.

The last column shows that heuristic H takes its maximum deviation of 8.62% when n=20. This figure decreases as n increases and takes its minimum value 1.52% when n=50.

Table 5.2 compares the performance of heuristics H_H, H_L and the best of these two heuristics for the different values of n.

Column 1 shows that for a given value of n, the average deviation of heuristic H_H is about one third of that of heuristic H (Table 5.1). A further substantial reduction in this average deviation was obtained when using heuristic H_L. Another further reduction was possible when choosing the best of H_H and H_L.

The second column of the same table shows that for heuristic H_H and n=20, 27 problems have optimum sequences (compared to six problems when heuristic H is used). This number is increased to 41 when heuristic H_L is used and to 42 when the best of H_H and H_L is chosen. The numbers of problems with optimum sequences decrease as n increases and reach their minimum values when n=50. These minimum values are 11, 18 and 22 when heuristics H_H, H_L and the best of H_H and H_L are used (compared to 0 when heuristic H is used).

The next five columns of the same table show all test problems of sizes 30, 40 and 50 have solutions within 3% of optimum when H_H is used, while using H_L leads to solutions to all problems (even for n=20) within 1% of optimum.

Column 8 shows that, for a given n, the maximum deviation from optimum is substantially smaller for H_L than it is for H_H.

The last column of the same table shows that for a given n, the number of cases H_L gave better results than H_H is much bigger than the number of cases H_H gave better results than H_L.

5.5 Concluding Remarks

In Section 5.2 we gave a full review of one machine heuristics. In Section 5.3 we proposed some new heuristics. Each of the proposed heuristics is written for a general problem and thus can be applied to all resulting special cases. Finally, in Section 5.4 we proposed a tree type heuristic. Here, a tree search procedure is considered and only one node is selected for branching within each level of the search tree. This node is chosen either because it has the smallest lower bound (this case is referred to as H_L) or because it has the smallest upper bound (this case is referred to as H_H).

The $1/r_i/\Sigma w_i C_i$ problem was considered to test the performance of heuristics H_L and H_H on test problems. The results showed both heuristics to perform reasonably well. The results also showed heuristic H_L to be substantially better than heuristic H_H which indicates that to obtain a near optimum solution, one should either use H_L or both H_L and H_H and choose the best solution obtained which appears to be a reasonable strategy.

Finally, there appears no reason why these tree type heuristics should not give results as good as obtained here when applied to other one machine scheduling problems. In fact, we see no reason why these heuristics cannot be used to obtain near optimum solutions for permutation flow-shop problems.

CHAPTER SIX

AN ALGORITHM FOR SINGLE MACHINE SEQUENCING WITH RELEASE DATES TO MINIMISE TOTAL WEIGHTED COMPLETION TIME

6.1 Introduction

The problem considered in this chapter may be stated as follows. Each of n jobs (numbered $1,\ldots,n$) is to be processed without interruption on a single machine which can handle only one job at a time. Job i ($i=1,\ldots,n$) becomes available for processing at its release date r_i, requires a processing time p_i and has a positive weight w_i. Given a processing order π of the jobs, the (earliest) completion time C_i for each job i can be computed. The objective is to find a processing order of the jobs which minimizes SWCT, the sum of weighted completion times $\Sigma w_i C_i$. The author acknowledges the substantial contributions of Dr. Potts to the development of this chapter.

When all release dates are equal, the problem can be solved using the algorithm of Smith (Smith, 1956) in which jobs are sequenced in non-increasing order of w_i/p_i. However, Lenstra et al. (Lenstra et al., 1977) have shown that when jobs have arbitrary release dates and unit weights the problem is NP-hard, which indicates that the existence of a polynomial bounded algorithm is unlikely. Consequently, branch and bound algorithms have been proposed for this problem with unit weights by Chandra (Chandra, 1979) and Dessouky & Deogun (Dessouky & Deogun, 1980). For the problem with arbitrary weights, Rinaldi & Sassano (Rinaldi & Sassano, 1977) have derived several dominance theorems. In this chapter a branch and bound algorithm for the problem with arbitrary weights is derived.

In Section 6.2 a heuristic method for sequencing the jobs is given. A lower bound, which is computed from this sequence, is derived in Section 6.3 and its working is demonstrated with a numerical example. An improvement

to the lower bound is presented in Section 6.4. Section 6.5 contains a statement of our first branching rule and gives some dominance rules which help to reduce the size of the search tree used in the branch and bound algorithm. A complete statement of the algorithm, including details of its implementation is given in Section 6.6. Our modified algorithm (including our second branching rule) is given in Section 6.7. Computational experience is presented in Section 6.8 which is followed by some concluding remarks in Section 6.9.

6.2 The Heuristic Method

It is well known that computation can be reduced by using a heuristic method to find a good solution to act as an upper bound on the sum of weighted completion times prior to the application of a branch and bound algorithm. Also, in our algorithm, a sequence generated by the heuristic method is used at each node of the search tree for calculating a lower bound.

The heuristic that is used has the property that the machine will never be kept unnecessarily idle. If there is a choice of jobs for the first unfilled position in the sequence which preserves this property, one with the largest w_i/p_i is chosen. A formal statement of the method is given below.

Step 1: Let S' be the set of all (unsequenced) jobs, let $H=0$ and $k=0$ and find $T' = \min_{j \in S'}\{r_j\}$.

Step 2: Find the set $S'' = \{j / j \in S', r_j \leq T'\}$ and find a job i with $i \in S''$ and with $w_i/p_i = \max_{j \in S''}\{w_j/p_j\}$.

Step 3: Set $k=k+1$, sequence job i in position k, set $T'=T'+p_i$, set $H=H+w_i T'$ and set $S' = S'-\{i\}$.

Step 4: If $S'=\emptyset$, then stop with the sequence generated having H as its sum of weighted completion times. Otherwise set $T' = \max\{T', \min_{j \in S'}\{r_j\}\}$ and go to Step 2.

It is possible to show that an upper bound on the worst-case performance of this heuristic does not exist. Consider the following two-job example with $r_1=0$, $p_1=h-2$, $w_1=1$, $r_2=1$, $p_2=1$ and $w_2=h$, where $h \geq 4$.

The heuristic H sequences job 1 before job 2 yielding $SWCT^H = h^2-2$. However, in the optimum sequence, job 2 is sequenced before job 1 yielding $SWCT^* = 3h$. Thus $SWCT^H/SWCT^* = (h^2-2)/3h$ which can be arbitrarily large.

We now derive sufficient conditions for the sequence generated by the heuristic to be optimum. However, some notation is introduced first. It is assumed that the jobs have been renumbered so that the sequence generated by the heuristic is $(1,\ldots,n)$ and the completion times of the jobs have been computed using $C_1 = r_1+p_1$, $C_i = \max\{r_i, C_{i-1}\} + p_i$ $(i=2,\ldots,n)$. The jobs may be partitioned into blocks $S_1,\ldots,S_k$ as follows. Job v_j is the *last job in a block* if $C_{v_j} \leq r_i$ for $i = v_j+1,\ldots,n$. A set of jobs $S_j = \{u_j,\ldots,v_j\}$ forms a *block* if the following conditions are satisfied:

(a) $u_j = 1$ or job u_j-1 is the last job in a block;

(b) job i is not the last job in a block for $i=u_j,\ldots,v_j-1$;

(c) job v_j is the last job in a block.

Job u_j is called the *first job in a block* and, for our heuristic, has the property that $r_{u_j} \leq r_i$ for $i=u_j+1,\ldots,n$. These definitions concerning blocks were proposed by Lageweg et al. (Lageweg et al., 1976).

The sufficient conditions for the sequence generated by the heuristic to be optimum are as follows.

Theorem 6.1

The sequence $(1,\ldots,n)$ generated by the heuristic is optimum if the jobs within each block S_j are sequenced in non-increasing order of w_i/p_i.

Proof

The result is first proved for the modified problem in which the release date of each job i in S_j is set to the release date of the first

job in block S_j $(j=1,\ldots,k)$. We first show that all jobs in block S_j should be sequenced before all jobs in block S_{j+1} $(j=1,\ldots,k-1)$ for this problem with reduced release dates. Consider any sequence and suppose that $i \varepsilon S_j$ is chosen so that i is as small as possible and so that job i is sequenced after a job in block $S_{j'}$ where $j' > j$. Suppose that this sequence is of the form $\sigma_1\sigma_2\sigma_3 i\sigma_4$, where σ_1 consists of all jobs in blocks $S_1,\ldots,S_{j-1}$, where σ_2 consists of jobs in block S_j and where the first job of σ_3 is a job in $S_{j'}$. Consider now the new sequence $\sigma_1\sigma_2 i\sigma_3\sigma_4$. The completion time of job i in this sequence is not greater than the release date of the first job in σ_3 which is in block $S_{j'}$ since the jobs in $\sigma_2 i$ are contained in block S_j. Thus the new sequence has a smaller sum of weighted completion times. Having established that, for an optimum sequence, all jobs within a block are sequenced in adjacent positions, their ordering is determined by Smith's rule. This proves the result for the problem with reduced release dates.

We now return to the original problem obtained by increasing the release dates to their initial values. Since this increase in release dates leaves the completion times unaltered, the sequence $(1,\ldots,n)$ is also optimum for the original problem.

It is seen in the next section that Theorem 6.1 is used in deriving our lower bound.

6.3 Derivation of the Lower Bound

The method used to obtain a lower bound is similar to the multiplier adjustment method proposed by Van Wassenhove (Van Wassenhove, 1979) for minimizing $\Sigma w_i C_i$ when jobs have zero release date and have deadlines. We obtain a lower bound by performing a Lagrangean relaxation of each release date constraint $C_i \geqslant r_i + p_i$ $(i=1,\ldots,n)$ after which it is replaced by a weaker constraint $C_i \geqslant r_i^* + p_i$ for some $r_i^* \leqslant r_i$. This yields the Lagrangean problem

$$L(\lambda) = \min\{\sum_{i=1}^{n} w_i C_i + \sum_{i=1}^{n} \lambda_i (r_i + p_i - C_i)\} \tag{6.1}$$

where $\lambda = (\lambda_1,\ldots,\lambda_n)$ is a vector of non-negative multipliers; the minimization is over all processing orders of the jobs with C_i $(i=1,\ldots,n)$ subject to machine capacity constraints and to the constraints $C_i \geq r_i^* + p_i$. We can write (6.1) as:

$$L(\lambda) = \min\{\sum_{i=1}^{n} w_i^* C_i\} + \sum_{i=1}^{n} \lambda_i (r_i + p_i)$$

where $w_i^* = w_i - \lambda_i$ $(i=1,\ldots,n)$. Thus, the Lagrangean problem is of the same form as the original problem but each job i has a new release date r_i^* and a new weight w_i^*. The choice of new release dates and of multipliers is discussed next. However, we shall restrict our choice of multipliers to the range $0 \leq \lambda_i \leq w_i$ $(i=1,\ldots,n)$ to ensure that $L(\lambda)$ does not become arbitrarily small. One possible approach is to set $r_i^* = 0$ so that the Lagrangean problem can be solved using Smith's rule. The value of λ which maximizes $L(\lambda)$ can then be found using the subgradient optimization method. However, this might entail much computation without guarantee of a tight lower bound. We prefer to retain the original values of the release dates, i.e. to set $r_i^* = r_i$ $(i=1,\ldots,n)$, but restrict the choice of multipliers so that the Lagrangean problem can be solved easily. This can be achieved by maximizing $L(\lambda)$ subject to the condition that the sequence generated by the heuristic solves the Lagrangean problem by yielding weights w_i^* $(i=1,\ldots,n)$ which satisfy the conditions of Theorem 6.1. Thus we require for each block S_j that

$$(w_i - \lambda_i)/p_i \leq (w_{i-1} - \lambda_{i-1})/p_{i-1} \text{ for } i = u_j+1,\ldots,v_j$$

It is clear that $L(\lambda)$ is maximized by choosing

$$\lambda_i = \begin{cases} 0 & \text{if } i = u_j, \\ \max\{0, w_i + (\lambda_{i-1} - w_{i-1}) p_i / p_{i-1}\} & \text{if } i = u_j+1, \ldots, v_j \end{cases} \quad (j=1,\ldots,k) \tag{6.2}$$

Having found C_i $(i=1,\ldots,n)$ using the sequence generated by the heuristic and λ_i $(i=1,\ldots,n)$ using (6.2), our lower bound can be written as

$$LB = \sum_{i=1}^{n} w_i C_i + \sum_{i=1}^{n} \lambda_i (r_i + p_i - C_i) \tag{6.3}$$

Example 6.1

The data for the example is summarized in the first three rows of Table 6.1. The jobs have already been renumbered so that the sequence generated by the heuristic method is (1,...,10).

Table 6.1: Data for the Example

i	1	2	3	4	5	6	7	8	9	10
r_i	1	62	93	146	206	223	230	271	219	219
p_i	50	41	37	28	60	19	97	37	76	94
w_i	10	3	8	8	3	6	10	3	6	6
C_i	51	103	140	174	266	285	382	419	495	589
λ_i	0	0	5.29	0	0	5	5.15	1.15	2.2	1.3
$\lambda_i(C_i-r_i-p_i)$	0	0	52.9	0	0	217	283	128	440	359

Having applied the heuristic method, the completion times of the jobs are computed. These are shown in row 4 of Table 6.1. The sum of weighted completion times is 17420. The blocks obtained from this sequence are $S_1 = \{1\}$, $S_2 = \{2,3\}$, $S_3 = \{4\}$ and $S_4 = \{5,6,7,8,9,10\}$. The multipliers,

obtained from (6.2), are shown in row 5 of Table 6.1. The value of the lower bound is computed from (6.3) using the bottom row of Table 6.1. This gives:

$$LB = 17420 - 1480 = 15940$$

6.4 The Improved Lower Bound

We assume that the multipliers defined in the previous section have been computed using (6.2). Suppose that the jobs are ordered within each block in non-decreasing order of multipliers to give a permutation $\pi = (\pi(1),\ldots,\pi(n))$ with the property that $S_j = \{\pi(u_j),\ldots,\pi(v_j)\}$ and that $\lambda_{\pi(u_j)} \leqslant \ldots \leqslant \lambda_{\pi(v_j)}$ $(j=1,\ldots,k)$. It is clear from (6.2) that $\lambda_{\pi(u_j)} = 0$ since the first job in a block always yields a zero multiplier. We now define

$$S_j^{(h)} = S_j^{(h-1)} - \{\pi(u_j+h-1)\} \quad (h=1,\ldots,v_j-u_j,\ j=1,\ldots,k)$$

where $S_j^{(0)} = S_j$ and

$$\mu^{(h)} = \lambda_{\pi(u_j+h)} - \lambda_{\pi(u_j+h-1)} \quad (h=1,\ldots,v_j-u_j,\ j=1,\ldots,k)$$

The set $S_j^{(h)}$ is obtained from the set $S_j^{(h-1)}$ by deleting a job having the smallest multiplier and $\mu_j^{(h)}$ is the difference in value between the multiplier of the job deleted and the smallest multiplier of the remaining jobs. From these definitions, we can rewrite (6.3) as

$$LB = \sum_{i=1}^{n} w_i C_i + \sum_{j=1}^{k} \sum_{h=1}^{v_j-u_j} \mu_j^{(h)} \left(b_j^{(h)} - \sum_{i\varepsilon S_j^{(h)}} C_i\right) \tag{6.4}$$

where $b_j^{(h)} = \Sigma_{i\varepsilon S_j^{(h)}} (r_i + p_i)$ $(h=1,\ldots,v_j-u_j,\ j=1,\ldots,k)$. (It is assumed that any summation is zero when its lower limit exceeds its upper limit). Clearly $b_j^{(h)}$ is a lower bound on $\Sigma_{i\varepsilon S_j^{(h)}} C_i$. However, if a better lower bound can be found, it is possible to increase LB. To obtain the best

possible bound on the sum of completion times of jobs having release dates would require the solution of an NP-hard problem (Lenstra et al., 1977). Since this is computationally expensive, we prefer to obtain a lower bound on $\Sigma_{i\varepsilon S_j^{(h)}} C_i$ by solving the corresponding pre-emptive scheduling problem in which the processing of any job can be interrupted and resumed at a later time. The pre-emptive problem is solved by using the procedure to be given below. The basis of this procedure is as follows. At any time when a job is completed or when a new job becomes available for processing, the job which is processed next is one with the shortest remaining processing time. If $\beta_j^{(h)}$ denotes the sum of completion times for the jobs in $S_j^{(h)}$ when they are sequenced using this shortest remaining processing time rule, we have the following improved lower bound:

$$LB' = LB + \sum_{j=1}^{k} \sum_{h=1}^{v_j-u_j} \mu_j^{(h)} (\beta_j^{(h)} - b_j^{(h)})$$

Since $\beta_j^{(h)} \geq b_j^{(h)}$, it is clear that $LB' \geq LB$.

Procedure for the 1/pmtn, $r_i/\Sigma C_i$ problem

Step 1: Let S' be the set of all jobs (i.e. $S' = \{1,\ldots,n\}$), f=0 and find $T' = \min_{i\varepsilon S'} \{r_i\}$. Set $p_i' = p_i$ for all i.

Step 2: Find the set $S'' = \{j/j\varepsilon S' \ \& \ r_j \leq T'\}$ and find a job $i\varepsilon S''$ with the smallest p_i'. This job or part of it is to be sequenced next.

Step 3: Find $t = \min_{j\varepsilon \bar{S}''} \{r_j\}$, where $\bar{S}'' = \{j/j\varepsilon S' \ \& \ j\notin S''\}$. If $\bar{S}''=\emptyset$, set $t=\infty$.

Step 4: Sequence p units of job i (from Step 2) next, where p = min $(p_i', t-T')$.

Step 5: Set $T'=T'+p$ and $p_i' = p_i' - p$.

Step 6: If $p_i' > 0$, go to Step 2.

Step 7: Set $S' = S' - \{i\}$ and $f = f + T'$. If S Ø stop with the optimum sequence generated having f as its sum of completion times; otherwise set $T' = \max(T', \min_{j \in S'} (r_j))$ and go to Step 2.

Theorem 6.2 (Conway et al., 1967)

The schedule obtained using the above procedure is an optimal solution to the $1/pmtn, r_i/\Sigma C_i$ *problem.*

With respect to the above procedure, we have the following theorem which is of some computational use. Here, jobs are assumed to have been sequenced in an increasing order of r_j, in case of ties job i with the shortest remaining processing time is sequenced first. Jobs are renumbered $\{1, \ldots, n\}$.

Theorem 6.3

Suppose that t is the completion time of an initial partial schedule. Suppose also that part of job i is processed optimally (using the above procedure) in an interval $[t, t_1]$ *(i.e. job i is not completed at time* t_1*). Also, suppose that there exists a job j with j chosen as small as possible so that* $r_j = t_1$*, then either job i or job j is processed in interval* $[t_1, r]$ *(if not completed before time r), where* $r = \min_{k \in S'} (r_k / r_k > t_1)$.

Proof

The existence of a job h with $r_h < t_1$ and $p_h < \min(p_i, p_j)$ contradicts the optimality of the above procedure. Also, the existence of a job h with $r_h = t_1$ and $p_h < \min(p_i, p_j)$ contradicts the choice of job j in the theorem.

Theorem 6.3 can be used in Step 6 of the above procedure; in the case when $p_i' > 0$ to determine whether to continue processing another part of job i or to start processing job j with $r_j = t$.

Example 6.2

In this example we shall explain how to compute our improved lower bound. Consider example 6.1 again. Ordering the jobs within each block

in a non-decreasing order of multipliers gives a permutation (1,2,3,4,5, 8,10,9,6,7). We now compute:

$$S_1^{(0)} = \{1\}$$

$$S_2^{(0)} = \{2,3\}, \quad S_2^{(1)} = \{3\} \quad \mu_2^{(1)} = 5.29$$

$$S_3^{(0)} = \{4\}$$

$$S_4^{(0)} = \{5,8,10,9,6,7\}, \quad S_4^{(1)} = \{8,10,9,6,7\} \quad \mu_4^{(1)} = 1.15$$

$$S_4^{(2)} = \{10,9,6,7\} \quad \mu_4^{(2)} = 1.3 - 1.15 = 0.15,$$

$$S_4^{(3)} = \{9,6,7\} \quad \mu_4^{(3)} = 0.9.$$

$$S_4^{(4)} = \{6,7\} \quad \mu_4^{(4)} = 2.85, \quad S_4^{(5)} = \{7\} \quad \mu_4^{(5)} = 0.1.$$

Clearly, we have $b_2^{(1)} = \beta_2^{(1)} = 130$. Solving the pre-emptive scheduling problem for jobs in $S_4^{(1)}$ (following the procedure above) we have:

Table 6.2*

J^i	9^1	6^1	6^2	9^2	8^1	9^3	10^1	7^1
P_i	76	19	19	76	37	76	94	97
P'_{J^i}	72	12	0	43	0	0	0	0
T_{J^i}	223	230	242	271	308	351	445	542

* J^i the ith part of job J.

P'_{J^i} the remaining processing time of job J after its ith part has been processed.

T_{J^i} the completion time of part J^i.

We can add the following constraint:

$$C_6 + C_7 + C_8 + C_9 + C_{10} \geq \beta_4^{(1)}$$

where $\beta_4^{(1)} = 242 + 308 + 351 + 445 + 542 = 1888$.

Thus we have $\beta_4^{(1)} = 1888$ compared with $b_4^{(1)} = 1485$. Similarly, $\beta_4^{(2)} = 1469$ with $b_4^{(2)} = 1177$, $\beta_4^{(3)} = 967$ with $b_4^{(3)} = 864$, $\beta_4^{(4)} = 581$ with $b_4^{(4)} = 569$, $\beta_4^{(5)} = 327$ with $b_4^{(5)} = 327$.

Thus $LB' = LB + 634 = 16574$.

6.5 Dominance Rules

If it can be shown that an optimum solution can always be generated without branching from a particular node of the search tree, then that node is dominated and can be eliminated. Dominance rules usually specify whether a node can be eliminated before its lower bound is calculated. Clearly, dominance rules are particularly useful when a node can be eliminated which has a lower bound that is less than the optimum solution. Nodes at level h of the search tree formed using our forwards branching rule FB represent initial partial sequences in which jobs in the first h positions have been fixed. The merits of this branching rule are discussed in the next section. The following results will show when any of the immediate successors of the node corresponding to an initial partial sequence σ are dominated. We assume that $\sigma = \sigma_1 h$, whenever σ is not empty. Also we define S to be the set of jobs not sequenced in σ and we define the earliest start time of these unsequenced jobs as $T = \max\{C(\sigma), \min_{i \in S}\{r_i\}\}$, where $C(\sigma)$ is the completion time of the last job of the partial sequence σ.

The first of our dominance theorems is a result of Rinaldi & Sassano (Rinaldi & Sassano, 1977). For completeness the proof is outlined.

Theorem 6.4 (Rinaldi & Sassano, 1977)

If job i is chosen with $i \in S$ and with $w_i/p_i = max_{j \in S}\{w_j/p_j\}$ and if $max\{r_i, T\} \leq max\{r_j, T\}$ for any $j \in S$, where $j \neq i$, then σj is dominated.

Proof

Consider any sequence $\sigma j \sigma' i \sigma''$ having σj as initial partial sequence. Job i can be interchanged with the job sequenced immediately before it without increasing the sum of weighted completion times. After the repeated application of this process, the sequence $\sigma i j \sigma' \sigma''$ will result which does not have σj as an initial partial sequence.

If, in Theorem 6.4, we have $r_i \leq T$, then the node corresponding to σ will have only one immediate successor σi. The lower bound for this successor is identical with that of its parent node and need not be computed again.

The next result is due to Dessouky & Deogun (Dessouky & Deogun, 1980). It states that the machine should not be kept idle throughout a time interval within which another job can be completely processed. Again, the proof is outlined.

Theorem 6.5 (Dessouky & Deogun, 1980)

If $r_j \geq C(\sigma i)$ for any $i, j \in S$, then σj is dominated.

Proof

Given any sequence $\sigma j \sigma' i \sigma''$ having σj as an initial partial sequence, a new sequence $\sigma i j \sigma' \sigma''$ can be formed in which job i has a smaller completion time and in which the jobs in σ' and σ'' do not have a larger completion time. This new sequence does not have σj as an initial partial sequence.

It is apparent that the conditions of Theorem 6.5 are most likely to be satisfied when job i is chosen with $C(\sigma i)$ as small as possible. It is expected that Theorem 6.5 will be most effective at reducing the size of the search tree when release dates have a large range.

Our final result is a consequence of dynamic programming. If the final two jobs of a partial sequence can be interchanged without increasing the sum of weighted completion times of jobs in the partial sequence and without increasing the time at which the machine becomes available to process the next unsequenced job, then this partial sequence is dominated. The importance of this type of dominance rule is often overlooked in single machine sequencing. Recalling that $\sigma = \sigma_1 h$, our dominance theorem is as follows.

Theorem 6.6

If $C(\sigma_1 jh) \leq C(\sigma_1 hj)$ and if $w_j C(\sigma_1 j) + w_h C(\sigma_1 jh) \leq w_h C(\sigma_1 h) + w_j C(\sigma_1 hj)$ for any $j \in S$, then $\sigma_1 hj$ is dominated.

Care must be taken when both of the conditions of Theorem 6.6 hold with equality that only one of the partial sequences $\sigma_1 hj$ and $\sigma_1 jh$ is discarded. It is possible to derive other dynamic programming dominance conditions involving the interchange of another pair of jobs or involving a larger group of jobs, but they are unlikely to be very effective once the three other theorems have been applied.

The dominance rules given in this section can only be used if the branching rules described in the following section is used. In Section 6.7.5 we shall propose a different method for branching.

6.6 The Algorithm

The branching rule FB (Forwards Branching) is discussed first. As was stated in the previous section, a node at level h of the search tree corresponds to an initial partial sequence in which jobs in the first h positions are fixed. This procedure has the advantage that once a job has been sequenced, its completion time is immediately computed and it can be discarded from consideration in all successor nodes. Alternatively, if

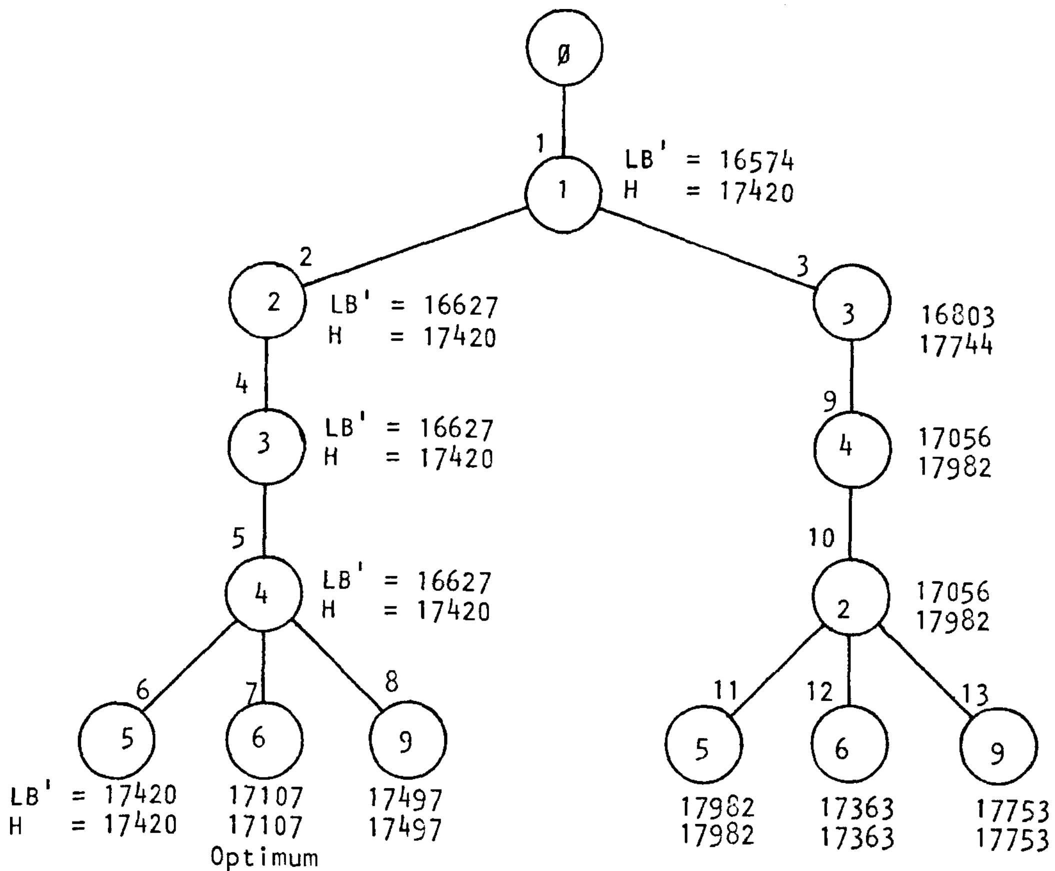
nodes correspond to final partial sequences, completion times of sequenced jobs depend on the processing order of unsequenced jobs. Before any new node is created, the dominance rules of the previous section are checked. If job i can be found satisfying the conditions of Theorem 6.4 with $r_i \leq T$, then a single successor node is created whose lower bound is the same as that of its parent. In other cases, as many nodes as possible are eliminated using Theorem 6.4. Then a job i is found with $C(\sigma i)$ as small as possible and the remaining nodes are checked for dominance using Theorem 6.5. Theorem 6.6 is applied to all nodes which have not been eliminated.

For each node of the search tree which cannot be eliminated by dominance rules, a lower bound is calculated. Firstly, the release date of each unsequenced job i is adjusted by setting $r_i = \max(r_i, T)$, where T denotes the earliest start time of unsequenced jobs. Then the heuristic method described in Section 6.2 and the lower bounding methods described in Section 6.3 and Section 6.4 are applied to the unsequenced jobs and the contributions of sequenced jobs are added. At level h of the search tree where there are $\bar{h} = n-h$ unsequenced jobs, the heuristic requires $O(\bar{h} \log \bar{h})$ steps. A further $\bar{h}$ steps are required to compute LB. If LB exceeds the value of a solution already computed, then this node is discarded. Otherwise, the lower bound LB' is computed. Since the solution of a pre-emptive scheduling problem with $\bar{h}$ jobs requires $O(\bar{h} \log \bar{h})$ steps, a further $O(\bar{h}^2 \log \bar{h})$ steps are required to solve the $O(\bar{h})$ pre-emptive scheduling problems. To summarise, LB requires $O(\bar{h} \log \bar{h})$ steps and LB' requires $O(\bar{h}^2 \log \bar{h})$ steps.

Finally, our search strategy is given. A newest active node search is used which selects a node from which to branch which has the smallest lower bound amongst nodes in the most recently created subset.

The full search tree for Example 6.1 using branching rule FB is given in Figure 6.1.

Figure 6.1: Search Tree for Example 6.1 (Using FB)



Key: Job's number is given inside each node while node's number is given above that node. The improved lower bound and $SWCT^H$ (Section 6.2) are denoted by LB' and H respectively.

6.7 Modified Algorithm

6.7.1 Branching

It is clear from the previous sections that a heuristic is used to obtain a sequence which is used in computing our lower bound LB. This sequence can be partitioned into blocks each of which consists of at least one job. The sharpness of the lower bound is determined according to the order in which jobs, within each block, are sequenced: if jobs in each block are sequenced in a non-increasing order of w_i/p_i ratios, then the lower and upper bounds computed using this sequence are equal.

In general, the gap between the lower and upper bounds computed using a particular sequence depends on how often this ordering (i.e. according to w_i/p_i ratios) is violated. If, for a sequence obtained at the top of the search tree, this ordering is violated among the first few jobs, then its effect on the lower bound will be reduced once we start branching from the beginning (i.e. using FB). Unfortunately, branching rule FB does not have a great effect on the gap between LB and UB if the ordering (i.e. according to w_i/p_i ratios) is violated somewhere deep in the sequence.

For this reason we have decided to use an approach based on selecting certain pairs of jobs i and j and deciding, at the top of the search tree, an ordering between the jobs of each pair. Each of these decisions (i.e. i before j or i after j) will be referred to as a *binary branching*. The conditions under which each pair of jobs is chosen to form two binary branchings will be given in Section 6.7.5.

The idea behind our binary branchings is that when solving the resulting problem, with parallel 1-level trees, a job i with small w_i/p_i together with other jobs will be replaced by a single composite job K having a much larger w_k/p_k.

A binary branching which corresponds to sequencing job i before job j, where job i is sequenced before job j in the sequence obtained for the parent node, will be referred to as a *left* branching, while the other binary branching (i.e. j before i) will be referred to as a *right* branching.

For a right branching, the precedence constraint is ignored and release date is adjusted such that j must be sequenced before i implies $r_j = \max\{r_j, r_i+p_i\}$. Ignoring the precedence constraint corresponding to a right branching is done for two reasons: The first reason is to make sure that the resulting precedence graph is series parallel. The second reason is to make the resulting problem easier to handle.

The resulting precedence graph (i.e. when applying the conditions of Sections 6.7.5 to select each pair of jobs and ignoring all right branchings) consists of parallel 1-level trees. One node of this graph is in series with all other nodes. This node will be referred to as the *root node*. All other nodes are in parallel with each other. Figure 6.2 shows a 1-level tree consisting of L nodes.

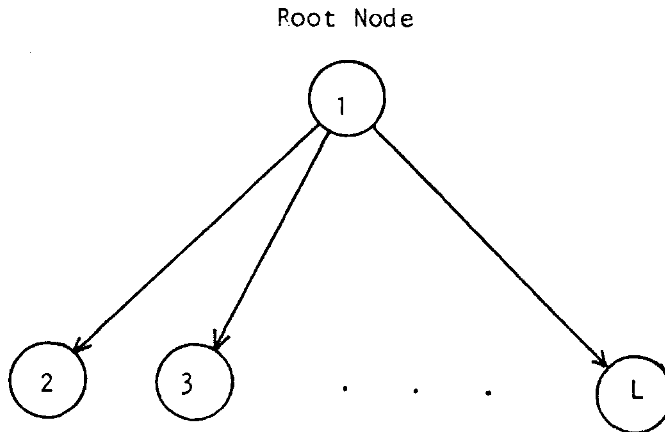
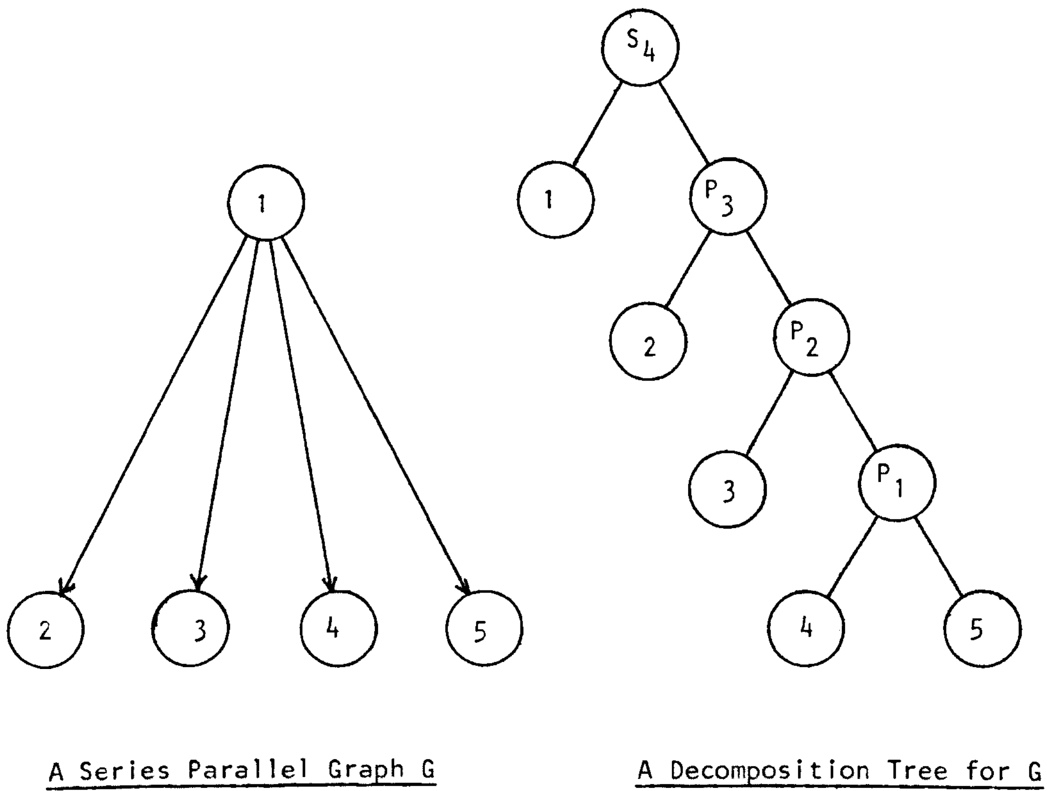


Figure 6.2

Minimizing $\Sigma w_i C_i$ subject to these precedence constraints can be done by using Lawler's O(n log n) series parallel algorithm (Lawler, 1978). This algorithm assumes that a decomposition tree is already known for the precedence constraints. To determine whether a given graph is series parallel and, if it is, to obtain a decomposition tree, one can use the method given in (Lawler, Tarjan & Valdez, ——) which requires O(n+m) steps where m is the number of arcs in the graph.

The method of Lawler, Tarjan and Valdez is based on repeatedly decomposing the precedence graph G into series and parallel components, so as to show how the transitive closure of G is obtained by rules (a) and (b) of Section 2.2.2 (i.e. conditions of series parallel graphs). "The result is a rooted binary tree we call *decomposition tree*. Each leaf of the

decomposition tree is identified with a node of G. Each internal node marked "S" indicates the series composition of subgraphs identified with its sons, with convention that the left son precedes the right son. Each internal node marked "P" indicates the parallel composition of the subgraphs identified with its sons. (Here the left-right ordering of sons is unimportant)." (Lawler, Tarjan & Valdez, ___). Figure 6.3 shows a graph G with its decomposition tree.



A Series Parallel Graph G   A Decomposition Tree for G

Figure 6.3

Given a decomposition tree, Lawler's algorithm works from the bottom of this decomposition tree upwards, finding an optimal sequence for a module M from the previously determined optimal sequences for its sons, M_1 and M_2.

We point out that if certain conditions are satisfied for two or more jobs, Lawler's algorithm replaces these jobs by a single *composite* job. The weight and the processing time of this composite job are set equal to the sum of the weights and the sum of the processing times of its component jobs respectively. The composite job is then treated as a single job. We also point out that Lawler's algorithm sequence independent jobs in a non-increasing order of $\rho(i) = w_i/p_i$ ratios.

"For parallel composition of M_1 and M_2, all that is necessary is to form the union of the two sets M_1 and M_2. Non-increasing ratio order is feasible and optimal for $M = M_1 \cup M_2$, assuming this is true for M_1, M_2 individually.

For series composition of M_1 and M_2, we first find a minimum-ratio job i in M_1 and maximum-ratio job j in M_2. If $\rho(i) > \rho(j)$, all that is necessary is to form the union of the sets M_1 and M_2. Non-increasing ratio order is feasible and optimal for $M = M_1 \cup M_2$, assuming this is true for M_1, M_2 individually." (Lawler, 1978).

Now suppose $\rho(i) \leqslant \rho(j)$, "what we do is to remove i from M_1, j from M_2 and form a composite job $k = (i,j)$. (Note: either i and j, or both, may themselves be composite jobs. The job k represents a sequence formed by joining together the two sequences represented by i and j).

Now let us find the next minimal element i in M_1: If $\rho(i) \leqslant \rho(k)$, we remove i from M_1 and form a new composite job $k = (i,k)$. We continue in this way until either M_1 is empty or $\rho(i) > \rho(k)$. Then we find the next maximal element in M_2. If $\rho(k) > \rho(j)$, we can safely let $M = M_1 \cup M_2 \cup \{k\}$. Buf if $\rho(k) \leqslant \rho(j)$, we remove j from M_2 and form a new composite job $k = (k,j)$. At this point we start all over again with M_1, at the top of this paragraph." (Lawler, 1978).

As an example, consider the problem with precedence graph G given in Figure 6.3 and with processing times and weights as shown in the following table.

i	1	2	3	4	5
p_i	3	2	1	2	1
w_i	1	7	5	8	10

Since jobs 4 and 5 are independent, then the set P_1 contains these two jobs in a non-increasing order of w_i/p_i, i.e. $P_1 = \{5,4\}$. Similarly, we have $P_2 = \{5,3,4\}$ and $P_3 = \{5,3,4,2\}$.

The steps required to form S_4 are as follows. We have $M_1 = \{1\}$ and $M_2 = \{5,3,4,2\}$. Since $w_1/p_1 \leq w_5/p_5$ we remove job 1 from M_1 and job 5 from M_2 and form a composite job k = (1,5) with $p_k = 4$ and $w_k = 11$. Now, consider the next job in M_2, i.e. job 3, we have $w_k/p_k \leq w_3/p_3$ and thus job 3 is removed from M_2 and a new composite job is formed, i.e. k = (k,3) = (1,5,3), with $p_k = 5$ and $w_k = 16$. For the same reason, one can form a composite job k = (1,5,3,4) and a composite job (1,5,3,4,2). Thus

$$S_4 = \{(1,5,3,4,2)\}$$

The idea behind our binary branchings is to replace a job i with small w_i/p_i together with other jobs by a single composite job K having a much larger w_K/p_K. Multipliers must be chosen so that the jobs in each tree are sequenced in adjacent positions in the same order as in the heuristic.

6.7.2 Composite Jobs

Two jobs i and j sequenced in adjacent positions using H are said to form a composite job ij if $w_i/p_i < w_j/p_j$ and if a branch $i \rightarrow j$ has been formed. It is clear that this composite job has a processing time p_i+p_j and a weight w_i+w_j.

We shall make it clear when talking about our new branching that only job i can be a composite job, $i = \{i_1, \ldots, i_L\}$ say, in which case the binary branching we make is $i_1 \rightarrow j$ and not $i_L \rightarrow j$ as one might expect. If i is a composite job, a third condition is needed to form a branch $i \rightarrow j$, namely $w_{i_L}/p_{i_L} > w_j/p_j$. This is to make sure that jobs forming the composite job are (except the first of these jobs) in a non-increasing order of w_i/p_i because all the binary branches we make are between each of these jobs and job i_1 as shown in Figure 6.4, where $\rho_i = w_i/p_i$. Figure 6.4 shows one of the conditions of forming a composite job also. However, all conditions for forming a composite job are given in Section 6.7.5.

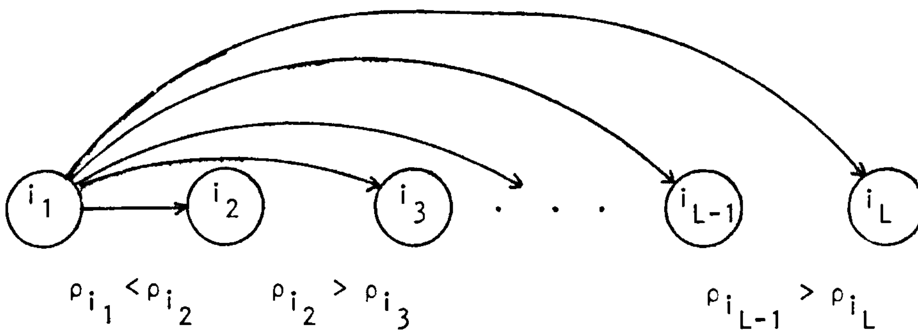


Figure 6.4: Forming a Composite Job

When dealing with a composite job instead of dealing with each of the jobs forming it separately, we are in fact dealing with a job with larger w_i/p_i. Hence, the effect a job i with small w_i/p_i and sequenced in the wrong position in π has on the lower bound will be reduced as soon as a composite job involving job i is formed. Composite jobs are particularly useful if they form the start of new blocks, in which case the multipliers for these composite jobs are reduced to zero.

6.7.3 Distributing the Multiplier of a Composite Job Amongst its Component Jobs

Consider a composite job K = {1,2,...,L} say, with a multiplier λ'_K, where:

$$\lambda'_K = \sum_{i=1}^{L} \lambda_i$$

and λ_i as given in (6.2). Recall from Section 6.3 that $0 \leq \lambda_i \leq w_i$ (i=1, ...,n). Dividing λ'_K among the L jobs forming the composite job K, we have: $\lambda'_1, \lambda'_2, \ldots, \lambda'_L$, where

$$\lambda'_K = \sum_{i=1}^{L} \lambda'_i$$

If L=2, then for λ'_1 and λ'_2 to be valid we must have

$$\frac{w_1 - \lambda'_1}{p_1} \leq \frac{w_2 - \lambda'_2}{p_2} \qquad (6.5)$$

If, on the other hand, L>2, then for $\lambda'_1, \ldots, \lambda'_L$ to be valid, the following inequalities must be satisfied also:

$$\frac{w_1 + w_2 - \lambda'_1 - \lambda'_2}{p_1 + p_2} \leq \frac{w_3 - \lambda'_3}{p_3} \qquad (6.6)$$

$$\frac{w_2 - \lambda'_2}{p_2} \geq \frac{w_3 - \lambda'_3}{p_3} \qquad (6.7)$$

$$\vdots$$

$$\frac{w_1 + w_2 + \ldots + w_{i-1} - (\lambda'_1 + \lambda'_2 + \ldots + \lambda'_{i-1})}{p_1 + p_2 + \ldots + p_{i-1}} \leq \frac{w_i - \lambda'_i}{p_i} \qquad (6.8)$$

$$\frac{w_{i-1} - \lambda'_{i-1}}{p_{i-1}} \geq \frac{w_i - \lambda'_i}{p_i} \qquad (6.9)$$

$$\vdots$$

$$\frac{w_1 + w_2 + \ldots + w_{L-1} - (\lambda_1' + \lambda_2' + \ldots + \lambda_{L-1}')}{p_1 + p_2 + \ldots + p_{L-1}} \leqslant \frac{w_L - \lambda_L'}{p_L} \qquad (6.10)$$

$$\frac{w_{L-1} - \lambda_{L-1}'}{p_{L-1}} \geqslant \frac{w_L - \lambda_L'}{p_L} \qquad (6.11)$$

It is clear that only inequalities (6.10) and (6.11) involve λ_L', and that only inequality (6.10) requires an upper bound on the value of λ_L'. This inequality can be written as follows:

$$\lambda_L' \leqslant \frac{1}{\sum_{i=1}^{L} p_i} \left(w_L \sum_{i=1}^{L-1} p_i - p_L \left(\sum_{i=1}^{L-1} w_i - \lambda_K'\right)\right)$$

Thus, we choose

$$\lambda_L' = \min\left(\lambda_K', \frac{1}{\sum_{i=1}^{L} p_i} \left(w_L \sum_{i=1}^{L-1} p_i - p_L \left(\sum_{i=1}^{L-1} w_i - \lambda_K'\right)\right)\right) \qquad (6.12)$$

If $\lambda_L' = \lambda_K'$ stop; otherwise an upper bound on the value of λ_{L-1}' has to be obtained to satisfy inequality (6.11).

$$p_L(w_{L-1} - \lambda_{L-1}') \geqslant p_{L-1}(w_L - \lambda_L')$$

$$\lambda_{L-1}' \leqslant w_{L-1} - \frac{p_{L-1}}{p_L}(w_L - \lambda_L') \qquad (6.13)$$

A second upper bound on the value of λ_{L-1}' can be obtained by applying equation (6.12) after setting $\lambda_K' = \lambda_K' - \lambda_L'$ and $L = L-1$, as we shall show in the following section.

6.7.4 Procedure

Distributing λ_K' of a composite job K amongst its component jobs.

Here we shall be interested in giving a procedure for distributing λ_K' among jobs forming the composite job $K = \{1,2,\ldots,L\}$.

Step 1: Set $\lambda'_{max} = \infty$.

Step 2: Find $\lambda'_L = \min\{(\lambda'_K, \lambda'_{max}, \frac{1}{\sum_{i=1}^{L} p_i}(w_L \sum_{i=1}^{L-1} p_i - p_L (\sum_{i=1}^{L-1} w_i - \lambda_K))\}$

Step 3: Calculate $\lambda'_{max} = w_{L-1} - \frac{p_{L-1}}{p_L}(W_L - \lambda'_L)$.

Step 4: If $\lambda'_{max} \leq \lambda'_K - \lambda'_L$, set $\lambda'_{L-1} = \lambda'_{max}$, $\lambda'_i = 0$, for all $i=1,\ldots,L-2$ and stop. Otherwise, set $\lambda'_K = \lambda'_K - \lambda'_L$, $L = L-1$ and go to Step 2.

6.7.5 Implementation of the Mixed Branching (MB)

Here we start by performing all possible binary branchings under the conditions to be given below. Once all allowable branchings have been performed we start sequencing the jobs one by one from the beginning using FB.

The advantage of our mixed branching is that besides being able to form composite jobs in the relaxed problem when computing the lower bound, we also have the advantage of using the dominance theorems of Section 6.5 which can only be applied when we start sequencing jobs from the beginning.

Procedure: Binary Branching (BB)

Here we shall give a procedure to select pairs of jobs to form binary branchings. Each selected pair of jobs i and j will form two binary branchings, namely i is constrained to be sequenced before j and i is constrained to be sequenced after j. The procedure is a function of three parameters I, M and Y. Parameters I and M are integers while parameter Y is real. The three parameters are determined by the researcher. Let $\pi = \{1,\ldots,n\}$ be the sequence obtained using the heuristic of Section 6.2.

1. Perform not more than M binary branchings.
2. Find a job i such that:
   - (a) Job i is the only job that can be sequenced in that position.
   - (b) The multiplier for job i is zero.

2.(c) Job i is not sequenced in the first I positions in π.

(d) If there is a choice, choose one that occurs first.

3. Find a job j which is not a composite job that is sequenced directly after i in π (i.e. j=i+1) such that:

(a) The multiplier for job j is positive.

(b) $w_i/p_i < w_j/p_j$.

(c) $\frac{w_i + w_j}{p_i + p_j} - \frac{w_i}{p_i} > Y$.

(d) If job i is a composite job, i.e. $i=(i_1,i_2,\ldots,i_L)$, the condition $w_{i_L}/p_{i_L} > w_j/p_j$ must be satisfied also.

We point out that if i is a composite job, i.e. $i=(i_1,\ldots,i_L)$, the two jobs to be selected to form two binary branchings are i_1 and j and not i_L and j as one would expect.

We also point out that we treat jobs single when applying the heuristic and that composite jobs remain composite (in the relaxed problem) until their root node is sequenced by FB.

Example 6.3

In this example we shall explain how our mixed branching procedure works. From example 6.1, we have:

Table 6.3

i	1	2	3	4	5	6	7	8	9	10
r_i	1	62	93	146	206	223	230	271	219	219
p_i	50	41	37	28	60	19	97	37	76	94
w_i	10	3	8	8	3	6	10	3	6	6
C_i	51	103	140	174	266	285	382	419	495	589
λ_i	0	0	5.29	0	0	5.05	5.15	1.15	2.2	1.3
No. of available jobs	1	1	1	1	1	4	4	3	2	1

Where job i is said to be available if $r_i \leq T'$ (T' is the completion time of the job sequenced in the previous position) or $r_i = \min_j\{r_j\}$ for all unsequenced jobs j.

Using the conditions of Section 6.7.5 we can perform the following binary branchings:

1(a). Forming $2 \to 3$

We can form a composite job (2-3) with

$p(2\text{-}3) = 78$

$w(2\text{-}3) = 11$

$\lambda(2\text{-}3) = 0$, since job (2-3) is the first job in the second block.

Thus $\lambda_2' = \lambda_3' = 0$.

$r_3 = \max\{r_3, r_2 + p_2\} = 103$.

Thus we have:

Table 6.4

i	1	(2-3)	4	5	6	7	8	9	10
r_i	1	62	146	206	223	230	271	219	219
C_i	51	140	174	266	285	382	419	495	589
λ_i	0	0	0	0	5.05	5.15	1.15	2.2	1.3
No. of available jobs	1	1	1	1	4	4	3	2	1

In which case LB = 15993 and LB' = 16627.

1(b). Forming $3 \to 2$

Since $w_3/p_3 > w_2/p_2$, we cannot form a composite job 3-2, we have $r_2 = 130$.

Table 6.5

i	1	3	2	4	5	6	7	8	9	10
r_i	1	93	130	146	206	223	230	271	219	219
C_i	51	130	171	199	266	285	382	419	495	589
λ_i	0	0	0	5.95	0	5.05	5.15	1.15	2.2	1.3
No. of available jobs	1	1	1	1	1	4	4	3	2	1

We have: LB = 16168 and LB' = 16803.

Since LB'(2-3) < LB'(3-2), we consider branch 2 → 3 first.

2(a). Forming 5 → 6

We can form a composite job (5-6) which has:

p(5-6) = 79

w(5-6) = 9

λ(5-6) = 0, since (5-6) is the first job in the fourth block.

From Table 6.4 we have:

Table 6.6

i	1	(2-3)	4	(5-6)	7	8	9	10
r_i	1	62	146	206	230	271	219	219
C_i	51	140	174	285	382	419	495	589
λ_i	0	0	0	0	0	0	0	0
No. of available jobs	1	1	1	1	4	3	2	1

We have LB = H = 17420. Thus the node corresponding to 5 → 6 is dead.

2(b). Forming $6 \rightarrow 5$

In this case, Table 6.4 becomes:

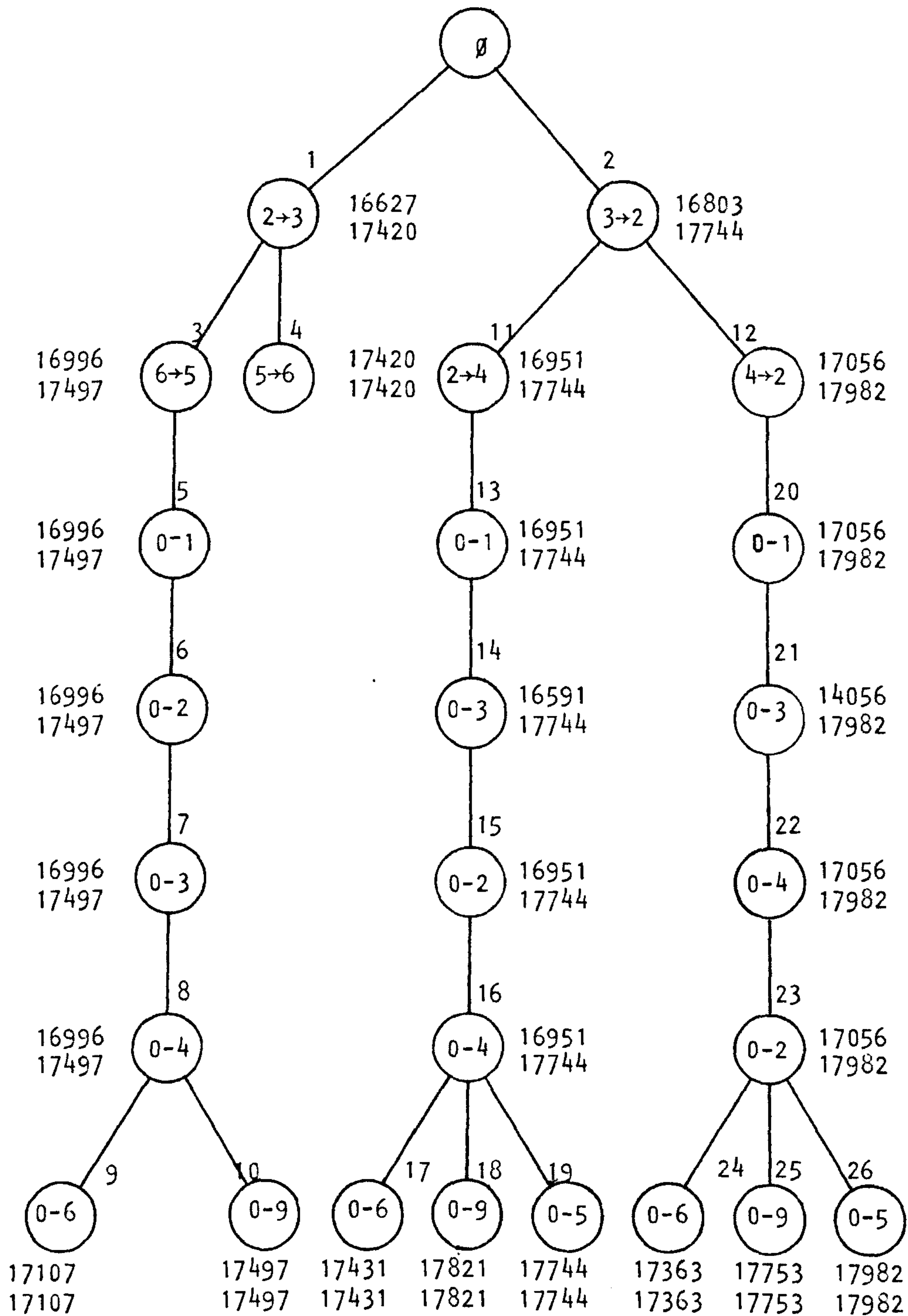
Table 6.7

i	1	(2-3)	4	9	6	7	8	10	5
r_i	1	62	146	219	223	230	271	219	242
C_i	51	140	174	295	314	411	448	542	602
λ_i	0	0	0	0	4.5	2.34	.08	0	0
No. of available jobs	1	1	1	2	5	4	3	2	1

We have LB = 16965 and LB' = 16996.

Since (according to Section 6.7.5) no other binary branchings can be performed, we start sequencing jobs one by one from the beginning using branching rule FB. The full search tree for this example is given in Figure 6.5.

Figure 6.1: Search Tree for Example 6.1



Key: A binary branching corresponding to job i being sequenced before job j is indicated by i→j (inside the node). A job i being sequenced first is indicated by 0-i. Number of node is given above each node. The other two numbers given outside each node are LB' (above) and $SWCT^H$ computed at that node.

6.8 Computational Experience

6.8.1 Test Problems

Our algorithms were tested on problems with 20, 30, 40 and 50 jobs. For each job i, an integer processing time p_i from the uniform distribution [1,100] and an integer weight w_i from the uniform distribution [1,10] were generated. Since the range of release dates is likely to influence the effectiveness of the algorithms, an integer release date for each job i was generated from the uniform distribution [0,50.5nR], where R controls the range of the distribution. The value 50.5n measures the expected total processing time. For each selected value of n, five problems were generated for each of the R values 0.2, 0.4, 0.6, 0.8, 1.0, 1.25, 1.5, 1.75, 2.0 and 3.0 producing fifty problems for each value of n.

6.8.2 Computational Results

The algorithms were coded in FORTRAN IV and run on a CDC 7600 computer.

Average computation times and average numbers of nodes are given in Table 6.8. Whenever a problem was not solved within the time limit of 60 seconds, computation was abandoned for that problem. Thus, in some cases the figures given in Table 6.8 will be lower bounds on the average computation times and the average numbers of nodes. Numbers of unsolved problems for the different values of R are listed in Table 6.9.

We start by discussing the results obtained using our first branching rule. It is clear from the average computation times that LB' is superior to LB. The difference in performance is most apparent for the thirty job problems. For n=40 and n=50 the true difference between LB and LB' in Table 6.8 is disguised by the unsolved problems. It can also be seen from Table 6.8 that the average computation time per node is considerably less for LB as is expected.

Table 6.8: Average Computation Times and Average Numbers of Nodes

n	FB and Lower Bound LB		FB and Lower Bound LB'		MB and Lower Bound LB'	
	Average Computation Time**	Average Number of Nodes	Average Computation Time**	Average Number of Nodes	Average Computation Time**	Average Number of Nodes
20	0.08	351	0.06	170	0.06	136
30	3.23*	10439*	1.47	2203	1.78	2380
40	17.30*	40991*	14.89*	24651*	14.81*	23591*
50	33.09*	65883*	30.58*	41255*	32.18*	38958*

** Times are in CPU seconds.
* Lower bounds because of unsolved problems.

Table 6.9: Numbers of Unsolved Problems

	n	R 0.2	0.4	0.6	0.8	1.0	1.25	1.5	1.75	2.0	3.0
FB & LB	20	0	0	0	0	0	0	0	0	0	0
	30	0	0	1	0	0	0	0	0	0	0
	40	0	0	3	4	1	2	0	0	0	0
	50	0	4	5	5	5	3	1	0	0	0
FB & LB'	20	0	0	0	0	0	0	0	0	0	0
	30	0	0	0	0	0	0	0	0	0	0
	40	0	0	2	3	1	1	0	0	0	0
	50	0	3	5	5	5	2	1	0	0	0
MB & LB'	20	0	0	0	0	0	0	0	0	0	0
	30	0	0	0	0	0	0	0	0	0	0
	40	0	0	2	2	1	0	0	0	0	0
	50	0	3	5	5	5	3	1	0	0	0

Table 6.9 shows that there are a total of 34 unsolved problems for LB compared with a total of 28 for LB' which again demonstrates the superiority of LB'. For both bounds, the problems with small R and large R are easiest. This is expected because for small R the release dates become unimportant once a few jobs have been sequenced enabling Theorem 6.4 to restrict the numbers of immediate successor nodes to one. However, when R is large the release dates become more important than the processing times and weights allowing Theorem 6.5 to successfully limit the size of the search tree. The hardest problems occur when R = 0.6, R = 0.8 and R = 1.0.

For the modified algorithm, initial experiments showed that the parameter values I = 15, Y = 0.046 and M = 200 to be a good choice. The results for these parameter values are shown in Tables 6.8 and 6.9, but they are not encouraging. This modified algorithm gave better results (than FB) only when n=40, in which case the number of unsolved problems is reduced by two (one each for R = 0.8 and R = 1.25). The worst results were obtained when n=50, in which case the number of unsolved problems is increased by one (R = 1.25). Obviously, choosing M = 0 would give the same results as FB.

The performance of our proposed bounds (using FB) was then tested on problems with up to 50 jobs where $w_i=1$, for $i=1,\ldots,n$ (i.e. the $1/r_i/\Sigma C_i$ problem). Average computation times and average number of nodes (for this special case) are given in Table 6.10. Number of unsolved problems for the different values of R are given in Table 6.11. The results show that both of our proposed bounds have not performed as well as they did for the case of general weights. This indicates that a different approach to solving this special case is required. Special purpose algorithms similar to that suggested by Dessouky and Deogun (Dessouky & Deogun, 1980) are likely to give better results.

Table 6.10: Average Computation Times and Average Numbers of Nodes for Problems with Unit Weights

n	FB and Lower Bound LB		FB and Lower Bound LB'	
	Average Computation Time**	Average Number of Nodes	Average Computation Time**	Average Number of Nodes
20	0.06	228	0.06	168
30	4.33*	15040*	4.21*	9323*
40	22.39*	55885*	20.23*	34284*
50	31.65*	64326*	30.85*	48006*

** Times are in CPU seconds.
* Lower bounds because of unsolved problems.

Table 6.11: Number of Unsolved Problems For Problems with Unit Weights

	n	R 0.2	0.4	0.6	0.8	1.0	1.25	1.5	1.75	2.0	3.0
LB	20	0	0	0	0	0	0	0	0	0	0
	30	0	0	0	1	0	1	0	0	0	0
	40	0	1	5	5	3	2	0	0	0	0
	50	0	4	5	5	5	2	0	0	0	0
LB'	20	0	0	0	0	0	0	0	0	0	0
	30	0	0	0	1	0	1	0	0	0	0
	40	0	0	3	5	3	1	0	0	0	0
	50	0	4	5	5	5	1	0	0	0	0

6.9 Concluding Remarks

The algorithm using the lower bound LB' is satisfactory for solving small and medium sized problems. However, a sharper lower bound is needed to cut down the size of the search tree when the number of jobs exceeds thirty.

One way in which the algorithm might be improved is to use the partitioning idea proposed by Rinaldi and Sassano (Rinaldi & Sassano, 1977). This states that if an optimum sequence σ of a subset of the original jobs can be found such that the release dates of all jobs not sequenced in σ are not less than the completion times of jobs in σ, then an optimum sequence to the complete problem exists which has σ as an initial partial sequence. When such a subsequence σ can be found, the remaining problem involving all jobs not sequenced in σ can be solved independently. However, the best way to find the necessary subset of jobs requires investigation.

The lower bounds LB and LB' are also valid lower bounds for the pre-emptive version of our problem. They could, with a suitable branching rule and with dominance rules, be used in a branch and bound algorithm for this pre-emptive scheduling problem which is NP-hard (Labetoulle et al., 1979). Our bounds can also be applied to the possibly more realistic non-pre-emptive problem in which unforced machine idle time is not allowed.

The modified algorithm proved to be useful when n=40. Altogether, its performance was not as effective as we hoped it would be. This was mainly due to the fact that different problems may need different sets of values of the parameters (I, Y and M). However, one way of increasing the efficiency of this modified algorithm might be by re-examining the conditions of Section 6.7.6 (i.e. the conditions of forming binary branchings).

As well as being of interest in its own right, the solution of the problem considered in this chapter might prove useful in obtaining lower bounds for flow-shop and job-shop problems based on Lagrangean relaxation. This seems to be worthy of future research.

CHAPTER SEVEN

THE SINGLE MACHINE PROBLEM WITH WEIGHTED SUM OF SQUARES OF COMPLETION TIMES

7.1 Introduction

Each of n jobs has to be processed without interruption on a single machine. The machine cannot process more than one job at a time. Each job i has a processing time p_i *and a positive weight* w_i*. Given any sequence of jobs the completion time* C_i *for any job i can be obtained assuming that processing starts at time zero. The objective is to find a sequence that minimizes the function* $f = \sum_{i=1}^{n} w_i C_i^2$.

Townsend (Townsend, 1978), to our knowledge, was the first to work on this problem. The problem is still open. However, Townsend pointed out that criteria for ordering jobs are unlikely to be simple since, in general, two jobs in adjacent positions cannot be ordered without reference to other jobs in the set. He illustrated this by giving the following three jobs example: $p_1 = 1$, $p_2 = 3$, $p_3 = 1$, $w_1 = 15$, $w_2 = 17$ and $w_3 = 7$, the optimum sequence is 123, but if p_1 is changed from 1 to 2, the optimum sequence is changed from 123 to 132. He also proposed a branch and bound procedure for solving this problem. The lower bound is based on ordering the jobs according to non-increasing w_i/p_i and making an adjustment to allow for the potential improvement that could be obtained by interchanging jobs i and j (for all i and j) if they are not in the right order according to non-increasing weights. Bagga and Kalra (Bagga & Kalra, 1980) studied this problem and proposed some elimination criteria to reduce the computation time.

In this chapter we propose a branch and bound algorithm to solve this problem. We start by giving some dominance theorems in Section 7.2. Townsend's lower bound is given in Section 7.3, followed by a new bounding

procedure in Section 7.4. The implementation of our new bounding procedure is given in Section 7.5, followed by a numerical example in Section 7.6. A full description of the algorithm is given in Section 7.7. The use of dominance rules is demonstrated in Section 7.8, followed by a discussion of a more general problem where precedence constraints among jobs exist in Section 7.9. Computational experiences are given in Section 7.10 followed by concluding remarks in Section 7.11.

7.2 Dominance Theorems

Suppose that T is the completion time of the last job in an initial partial sequence σ. And suppose that jobs i and j are to be sequenced in adjacent positions directly after all jobs in σ. Let f_{ij} and f_{ji} be the penalties associated with jobs i and j when they are order ij and ji respectively. Then we have:

$$f_{ij} = w_i(T + p_i)^2 + w_j(T + p_i + p_j)^2$$

and

$$f_{ji} = w_j(T + p_j)^2 + w_i(T + p_j + p_i)^2$$

Thus, $f_{ij} \leqslant f_{ji}$ if

$$w_j p_i(2T + p_i + 2p_j) \leqslant w_i p_j(2T + p_j + 2p_i)$$

or (7.1)

$$w_j p_i(2T + p_i + p_j) + w_j p_i p_j \leqslant w_i p_j(2T + p_i + p_j) + w_i p_i p_j$$

This leads to the following theorem:

Theorem 7.1 (Townsend, 1978)

There exists an optimal sequence in which job i is sequenced before job j if:

$$w_i/p_i \geqslant w_j/p_j \tag{7.2}$$

$$w_i \geqslant w_j \tag{7.3}$$

Corollary 7.1

There exists an optimal sequence in which job i precedes job j if $p_i \leq p_j$ *and* $w_i \geq w_j$.

Theorem 7.2

The $1/p_i \leq p_j \to w_i \geq w_j/\Sigma w_i C_i^2$ *(agreeable weights) problem can be solved by ordering the jobs in a non-increasing order of their weights. In case of ties, sequence job i with the shortest processing time first.*

Proof

Obvious.

Let π_1 and π_2 be two sequences obtained by ordering jobs according to inequalities 7.2 and 7.3 respectively. Break ties by sequencing jobs according to inequality 7.3 in the first case and inequality 7.2 in the second case. Jobs with equal processing times and equal weights are sequenced in the same order in both sequences.

Theorem 7.3 (Bagga & Kalra, 1980)

If the first r positions in π_1 *and* π_2 *contain the same jobs (need not be in the same order), then none of the jobs in the remaining n-r positions can occupy any of the first r positions in the optimum sequence.*

Corollary 7.2 (Bagga & Kalra, 1980)

If jobs in the first r positions (and/or jobs in the last n-r positions) are sequenced according to both inequalities (7.2 and 7.3) then these jobs will appear in the same order in the optimum sequence.

Corollary 7.3 (Bagga & Kalra, 1980)

If π_1 *and* π_2 *have the same permutation of jobs then that permutation is an optimal sequence.*

7.3 Townsend Lower Bound

It is clear from Theorem 7.1 and Corollary 7.3 that if inequalities 7.2 and 7.3 are satisfied for a particular sequence, then that sequence is

optimum. On the other hand, if inequality 7.2 only is met, a lower bound is obtained by making an adjustment to allow for the potential improvement that could be obtained by interchanging each pair of jobs i and j for which inequality 7.2 only is satisfied.

Thus, to obtain a lower bound, jobs are sequenced according to inequality 7.2 alone. Then it can be seen from inequality 7.1 that the maximum reduction in penalty that can occur when interchanging the order of two jobs from ij to ji is:

$$(w_j - w_i)\ p_i p_j$$

Thus, a lower bound is given by (assuming that jobs are renumbered 1,...,n):

$$LB_T = \sum_{i=1}^{n} w_i \left(\sum_{j=1}^{i} p_j \right)^2 - \sum_{i=1}^{n} \sum_{\substack{j=1 \\ \& \ w_j > w_i}}^{n} (w_j - w_i)\ p_i p_j$$

Example 7.1 (Townsend, 1978)

	1	2	3	4	5
p_i	10	4	6	1	2
w_i	2	5	7	3	1
w_i/p_i	1/5	5/4	7/6	3	1/2

Ordering the jobs in a non-increasing order of w_i/p_i we have π = 42351. To calculate a lower bound we need to consider the following interchanges.

51 to 15: with potential reduction = (2-1) x 10 x 2 = 20
23 to 32: with potential reduction = (7-5) x 4 x 6 = 48
43 to 34: with potential reduction = (7-3) x 6 x 1 = 24
42 to 24: with potential reduction = (5-3) x 4 x 1 = 8

Thus, from 7.5, a lower bound LB_T is given by:

$$LB_T = 2202 - (20 + 48 + 24 + 8) = 2102$$

If the branching procedure (a) of Section 3.2.2(i.e. sequencing the jobs one by one from the beginning), one can compute the following lower bounds in a similar way:

$$LB_T(4) = 2202 - (20 + 48) = 2134$$

$$LB_T(5) = f(54231) - (8 + 24 + 48)$$
$$= 2517 - 80 = 2437$$

The full search tree for this example is given in Figure 7.1.

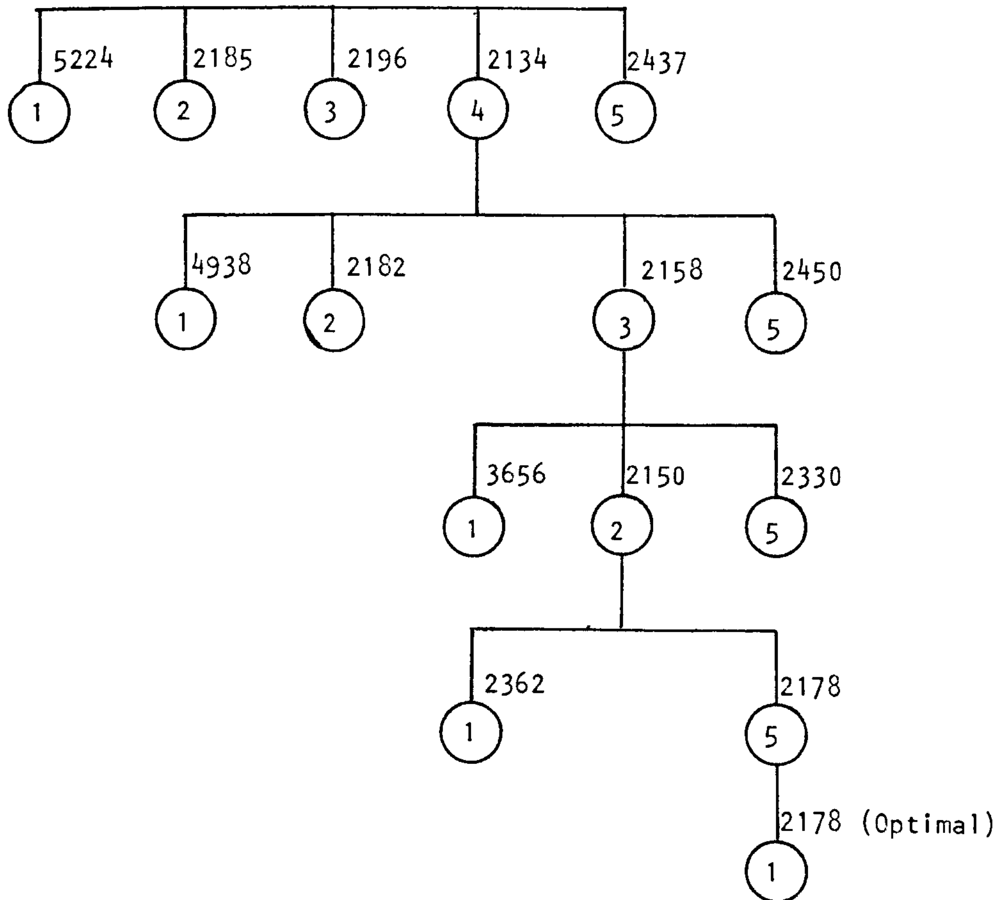


Figure 7.1: Search Tree for Example 7.1

Remark

Ordering the jobs according to inequalities 7.2 and 7.3, we have: $\pi_1 = 42351$ and $\pi_2 = 32415$. Since jobs 2, 3 and 4 are sequenced in the first three positions in both sequences, then jobs 1 and 5 cannot be sequenced in the first three positions of an optimal sequence and the search tree of Figure 7.1 is reduced to that given in Figure 7.2.

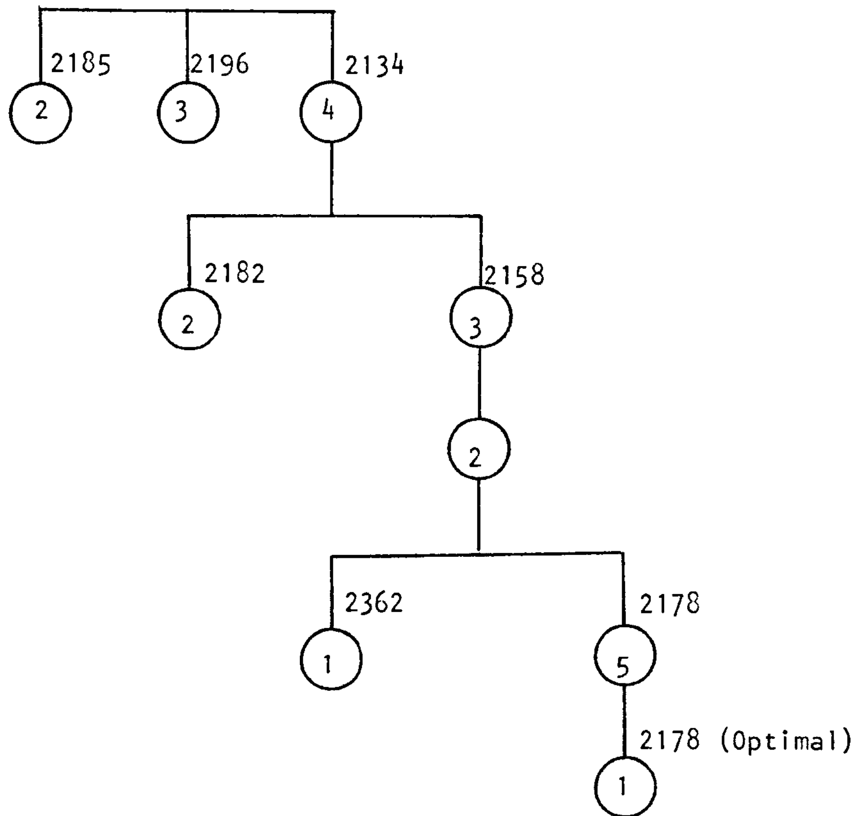


Figure 7.2: Reduced Search Tree for Example 7.1

7.4 New Bounding Procedure

The approach we are going to use to derive our bound is similar to that used by Balas and Christofides (Balas & Christofides, 1981) for the travelling salesman problem, and that of Potts (Potts, 1981) for the single machine sequencing with precedence constraints in which the objective is to minimize the weighted sum of completion times.

It will be useful at this stage to give the following definitions. Given a directed acyclic graph G, we define the transitive closure of G as the graph obtained by adding all arcs (i,j) to G whenever there is a directed path from vertex i to vertex j. We can also define the adjacency matrix of G as the $n \times n$ matrix $A = (a_{ij})$, where $a_{ij} = 1$ if an arc (i,j) exists in the transitive closure of G and $a_{ij} = 0$ otherwise.

To derive a lower bound, we start by formulating the problem as a zero one programming problem. We define a zero one variable x_{ik} (i,k=1, ...,n), where $x_{ii} = 1$ (i=1,...,n), as follows:

$$x_{ik} = \begin{cases} 1, & \text{if job } i \text{ is to be sequenced before job } k. \\ 0, & \text{otherwise.} \end{cases}$$

It is clear that the completion time of job k is given by $\sum_{i=1}^{n} p_i x_{ik}$ and hence the problem can be written as follows:

$$\text{Minimize} \quad \sum_{k=1}^{n} w_k C_k^2$$

$$= \sum_{k=1}^{n} w_k \left(\sum_{i=1}^{n} p_i x_{ik} \right)^2$$

$$= \sum_{k=1}^{n} w_k \left(\sum_{i=1}^{n} p_i^2 x_{ik} + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} p_i p_j x_{ik} x_{jk} \right) \qquad (7.4a)$$

Subject to:

$$x_{ij} + x_{ji} = 1, \qquad i,j=1,\ldots,n \;\&\; i \neq j \qquad (7.5)$$

$$x_{ij} + x_{jk} + x_{ki} \geq 1, \qquad i,j,k=1,\ldots,n \;\&\; i \neq j \neq k \neq i \qquad (7.6)$$

$$x_{ij} = 0 \text{ or } 1, \qquad i,j=1,\ldots, n \;\&\; i \neq j \qquad (7.7)$$

The constraint (7.5) ensures that any job i is to be sequenced either before or after another job j. The matrix $X = (x_{ij})$ can be regarded as

the adjacency matrix of a directed graph G_X. The constraints (7.6) ensure that G_X contains no cycles. When all constraints are satisfied G_X defines a complete ordering of the jobs.

The coefficient $p_i^2 w_k$ of x_{ik} can be regarded as the cost of scheduling job k not before job i and the coefficient $p_i p_j w_k$ of $x_{ik} x_{jk}$ can be regarded as the cost of scheduling job k not before jobs i and j. Hence, it is possible to introduce the 3-dimensional cost array $c = (c_{ijk})$, where $c_{ijk} = p_i p_j w_k$. It is clear that if $i \neq j \neq k \neq i$, then each of c_{ijk} and c_{jik} is a contribution to the cost of scheduling job k after the two jobs i and j and that c_{iik}, c_{ikk} and c_{kik} can be regarded as a contribution to the cost of scheduling job k after job i.

Now, the problem can be written as follows:

$$\text{Minimize} \quad \sum_{k=1}^{n} \left(\sum_{i=1}^{n} c_{iik} x_{ik} + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} c_{ijk} x_{ik} x_{jk} \right) \qquad (7.4b)$$

Subject to (7.5), (7.6) and (7.7).

We now introduce new zero one variables y_{ijk} and z_{ik}, where $y_{ijk} = 0$ if $i=j$, $i=k$ or $j=k$ and $z_{ii} = 0$ $(i=1,\ldots,n)$. Variables y_{ijk} and z_{ik} can be defined as follows:

$$y_{ijk} = \begin{cases} 1, & \text{if jobs } i \text{ and } j \text{ are not to be sequenced after job } k. \\ 0, & \text{otherwise.} \end{cases}$$

and

$$z_{ik} = \begin{cases} 1, & \text{if job } i \text{ is not to be sequenced after job } k. \\ 0, & \text{otherwise.} \end{cases}$$

In other words:

$$y_{ijk} = \begin{cases} x_{ik} x_{jk}, & \text{if } i \neq j \neq k \neq i. \\ 0, & \text{otherwise.} \end{cases}$$

and $z_{ik} = x_{ik}$ when $i \neq k$.

It is obvious that $y_{ijk} = y_{jik}$.

Problem(7.4b) can be written as follows:

$$\text{Minimize} \quad \sum_{k=1}^{n} \sum_{i=1}^{n} h_{ik} z_{ik} + \sum_{k=1}^{n} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ijk} y_{ijk} + \sum_{k=1}^{n} p_k^2 w_k \tag{7.8}$$

where $h_{ik} = c_{iik} + c_{ikk} + c_{kik}$

Subject to:

$$z_{ij} + z_{ji} = 1 \qquad i,j=1,\dots,n,\ i\neq j \tag{7.9}$$

$$y_{ijk} + y_{ikj} + y_{jki} = 1 \qquad i,j,k=1,\dots,n,\ i\neq j\neq k\neq i \tag{7.10}$$

$$z_{ij} + z_{jk} + z_{ki} \geq 1 \qquad i,j,k=1,\dots,n,\ i\neq j\neq k\neq i \tag{7.11}$$

$$y_{ijk} + z_{ki} + z_{kj} \geq 1 \qquad i,j,k=1,\dots,n,\ i\neq j\neq k\neq i \tag{7.12}$$

$$y_{ijk} + y_{kij} + z_{ki} \geq 1 \qquad i,j,k=1,\dots,n,\ i\neq j\neq k\neq i \tag{7.13}$$

$$y_{ijk} + z_{ki} + z_{ij} \geq 1 \qquad i,j,k=1,\dots,n,\ i\neq j\neq k\neq i \tag{7.14}$$

$$y_{ijk} + y_{kji} + z_{kj} \geq 1 \qquad i,j,k=1,\dots,n,\ i\neq j\neq k\neq i \tag{7.15}$$

$$y_{ijk} + z_{kj} + z_{ji} \geq 1 \qquad i,j,k=1,\dots,n,\ i\neq j\neq k\neq i \tag{7.16}$$

$$y_{ijk} + y_{kji} + z_{ij} \geq 1 \qquad i,j,k=1,\dots,n,\ i\neq j\neq k\neq i \tag{7.17}$$

$$y_{ijk} = 0,1 \ \&\ z_{ij} = 0,1 \qquad i,j,k=1,\dots,n \tag{7.18}$$

We define a directed graph $G_{Z,Y}$ which has two classes of nodes. Jobs $1,\dots,n$ form the first class of nodes while pairs of jobs ij, $i,j=1,\dots,n$ & $i\neq j$, form the second class of nodes (dummy nodes). An arc from node i to node j exists in $G_{Z,Y}$ if $z_{ij} > 0$, while an arc from node ij to node k exists in $G_{Z,Y}$ if $y_{ijk} > 0$.

Constraints (7.9),...,(7.18) form an essential part of the problem formulation. As constraint (7.5), constraint (7.9) ensures that any job i is to be sequenced either before or after another job j. Constraints (7.10) ensure that among any three jobs one job only can be sequenced last. Constraints (7.11),...,(7.17) ensure that the graph $G_{Z,Y}$ contains no cycles.

Constraints (7.11) say that any three jobs i, j and k will have job j sequenced after job i and (or) job k is sequenced after job j and (or) job i is sequenced after job k. Constraints (7.12) can be interpreted as follows: Any three jobs i, j and k will have job k sequenced after jobs i and j or job i sequenced after job k or (and) job j sequenced after job k. Constraints (7.13) say that any three jobs i, j and k will have job k sequenced last or job j sequenced last or (and) job i is sequenced after job k. All other constraints can be interpreted in a similar way. We point out that some of these constraints are redundant, but are included here for a reason which will become obvious at some later stage. It is possible to derive more general cycle elimination constraints:

$$\sum_{v=1}^{q} y_{ijk_v} + \sum_{v=q+1}^{r} z_{uk_v} \geq 1 \tag{7.19}$$

where $k_1,k_2,\ldots,k_r$ correspond to r ($r \geq 3$) different jobs and where $i,j,u \in \{k_1,\ldots,k_r\}$, $i \neq j \neq k_v \neq i$, $u \neq k_v$, $q=0,1,\ldots,r-1$ (we assume that $q=0$ implies that $\sum_{v=1}^{q} y_{ijk_v} = 0$).

Given i, j and k such that $i \neq j \neq k \neq i$, there are not more than eight constraints of type (7.10) since there are two ways to represent each variable (e.g. y_{ijk} is equivalent to y_{jik}). For the same reason there exist not more than four constraints of the type (7.13), (7.15) and (7.17).

Each of the constraints (7.9) may now be introduced into a Lagrangean dual, with associated multiplier μ_{ik} to give:

$$L^{o} = \sum_{k=1}^{n} \sum_{i=1}^{n} (h_{ik} - \mu_{ik} - \mu_{ki})\, z_{ik} + \sum_{k=1}^{n} \sum_{i=1}^{n} \mu_{ki} +$$

$$\sum_{k=1}^{n} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ijk}\, y_{ijk} + \sum_{k=1}^{n} p_k^2\, w_k$$

It should be noted that to retain symmetry two multipliers μ_{ik} and μ_{ki} are associated with every constraint of the type (7.9). For the coefficient of z_{ik} and z_{ki} to remain non-negative and the multipliers μ_{ik} and μ_{ki} to provide as large a contribution as possible to the lower bound, we set:

$$\mu_{ik} = \mu_{ki} = \tfrac{1}{2}\min(h_{ik}, h_{ki}), \quad i,j=1,\dots,n, i \neq j \tag{7.20}$$

We now define a reduced cost matrix $H^{(0)} = (h_{ij}^{(0)})$, where

$$h_{ik}^{(0)} = h_{ik} - \mu_{ik} - \mu_{ki}$$

Thus, according to (7.20) either $h_{ik}^{(0)} = 0$ or $h_{ki}^{(0)} = 0$.

We next introduce constraints of type (7.10) to the Lagrangean dual. Suppose that (s-1) of these constraints have been introduced into the Lagrangean dual with multipliers $\gamma^{(1)},\dots,\gamma^{(s-1)}$ to give:

$$L^{(s-1)} = \sum_{k=1}^{n}\sum_{i=1}^{n} h_{ik}^{(0)} z_{ik} + \sum_{k=1}^{n}\sum_{i=1}^{n} \mu_{ki} + \sum_{k=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n}$$

$$c_{ijk}^{(s-1)} y_{ijk} + \sum_{k=1}^{s-1} \gamma^{(k)} + \sum_{k=1}^{n} p_k^2 w_k \tag{7.21}$$

As mentioned before, there are not more than eight constraints of type (7.10) for every given value of i, j and k (because y_{ijk} is equivalent to y_{jik}, 2^3 constraints exist). One of these constraints, $y_{ijk} + y_{ikj} + y_{jki} = 1$, will be introduced into the Lagrangean dual below, each of the other constraints may be dealt with in a similar way.

Introducing the constraint $y_{ijk} + y_{ikj} + y_{jki} = 1$ into the Lagrangean dual gives:

$$L^{(s)} = L^{(s-1)} + \gamma^{(s)} (1 - y_{ijk} - y_{ikj} - y_{jki})$$

where $\gamma^{(s)} = \min\{c_{ijk}^{(s-1)}, c_{ikj}^{(s-1)}, c_{jki}^{(s-1)}\}$.

This choice of $\gamma^{(s)}$ will ensure that the coefficients of y_{ijk} in $L^{(s)}$ remain non-negative. If we update the 3-dimensional reduced cost array, $c^{(s)}$, using:

$$c_{ijk}^{(s)} = c_{ijk}^{(s-1)} - \gamma^{(s)}, \quad c_{ikj}^{(s)} = c_{ikj}^{(s-1)} - \gamma^{(s)} \quad \text{and}$$

$$c_{jki}^{(s)} = c_{jki}^{(s-1)} - \gamma^{(s)}$$

We can write $L^{(s)}$ in the following form:

$$L^{(s)} = \sum_{k=1}^{n} \sum_{i=1}^{n} h_{ik}^{(0)} z_{ik} + \sum_{k=1}^{n} \sum_{i=1}^{n} \mu_{ki} + \sum_{k=1}^{n} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ijk}^{(s)} y_{ijk}$$

$$+ \sum_{k=1}^{s} \gamma^{(k)} + \sum_{k=1}^{n} p_k^{\,2} w_k \tag{7.22}$$

Suppose that N constraints of type (7.10) are added, i.e. s=1,...,N. Let $D^{(0)} = (d_{ijk}^{(0)})$, where $d_{ijk}^{(0)} = c_{ijk}^{(N)}$, for all i,j,k=1,...,n.

We next consider the cycle elimination constraints. The general form (7.19) of these constraints will be assumed. Suppose that (t-1) of these constraints have been introduced to the Lagrangean dual with associated multipliers $\lambda^{(1)},\ldots,\lambda^{(t-1)}$ to give:

$$L^{(N+t-1)} = \sum_{k=1}^{n} \sum_{i=1}^{n} h_{ik}^{(t-1)} z_{ik} + \sum_{k=1}^{n} \sum_{i=1}^{n} \mu_{ki}$$

$$+ \sum_{k=1}^{t-1} \lambda^{(k)} + \sum_{k=1}^{n} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ijk}^{(t-1)} y_{ijk} + \sum_{k=1}^{n} \gamma^{(k)}$$

$$+ \sum_{k=1}^{n} p_k^{\,2} w_k \tag{7.23}$$

A new constraint of type (7.19) will now be introduced into the Lagrangean dual to give:

$$L^{(N+t)} = L^{(N+t-1)} + \lambda^{(t)} \left(1 - \sum_{v=1}^{q} y_{ijk_v} - \sum_{v=q+1}^{r} z_{uk_v}\right) \qquad (7.24)$$

where

$$\lambda^{(t)} = \min\{\min_{v=1,\dots,q} \{d_{ijk_v}\}, \min_{v=q+1,\dots,r} \{h_{uk_v}\}\}$$

It is clear that this choice of $\lambda^{(t)}$ will make sure that the coefficients of z_{ik} and y_{ijk} in $L^{(t)}$ remain non-negative.

$D^{(t)}$ and $H^{(t)}$ can be updated as follows:

$$D^{(t)} = D^{(t-1)} - \lambda^{(t)} E^{(t)}$$

where $E^{(t)} = (e^{(t)}_{ijk})$ and $G^{(t)} = (g^{(t)}_{ik})$ are two arrays defined as follows:

$$e^{(t)}_{ijk} = 1, \quad \text{if variable } y_{ijk} \text{ is in constraint (7.19) when added at step } t,$$
$$= 0, \quad \text{otherwise,}$$

and

$$g^{(t)}_{ik} = 1, \quad \text{if variable } z_{ik} \text{ is in constraint (7.19) when added at step } t,$$
$$= 0, \quad \text{otherwise.}$$

Having updated $D^{(t)}$ and $H^{(t)}$ it is possible to write $L^{(N+t)}$ in a form similar to (7.23):

$$L^{(N+t)} = \sum_{k=1}^{n} \sum_{i=1}^{n} h^{(t)}_{ik} z_{ik} + \sum_{k=1}^{n} \sum_{i=1}^{n} \mu_{ki} + \sum_{k=1}^{t} \lambda^{(t)} + \sum_{k=1}^{n} \sum_{i=1}^{n} \sum_{j=1}^{n} d^{(t)}_{ijk} y_{ijk} + \sum_{k=1}^{N} \gamma^{(k)} + \sum_{k=1}^{n} p_k^{\,2} w_k \qquad (7.25)$$

The validity of the proposed lower bound will now be proved in the following theorem.

Theorem 7.4

A lower bound for the problem is given by:

$$LB^{(N+t)} = \sum_{k=1}^{n} \sum_{i=1}^{n} \mu_{ki} + \sum_{k=1}^{N} \gamma^{(k)} + \sum_{k=1}^{t} \lambda^{(k)} + \sum_{k=1}^{n} p_k^2 w_k$$

Proof

From duality theorem, the minimum value of $L^{(N+t)}$ provides a lower bound. Now since $h_{ik}^{(t)} \geq 0$, $d_{ijk}^{(t)} \geq 0$, $z_{ik} \geq 0$ and $y_{ijk} \geq 0$, then $L^{(N+t)}$ is minimized by setting $z_{ik} = 0$ whenever $h_{ik}^{(t)} > 0$ and setting $y_{ijk} = 0$ whenever $d_{ijk}^{(t)} > 0$. This will yield $h_{ik}^{(t)} z_{ik} = 0$ and $d_{ijk}^{(t)} y_{ijk} = 0$ and hence yield the required lower bound.

It is clear that to increase the lower bound, as many constraints as possible are added. Each time a constraint is added, the lower bound is increased by the value of the multiplier and hence it is helpful to add constraints with positive multipliers only.

If $D^{(t)}$ and $H^{(t)}$ are fully reduced then it should be possible to find values of the variables y_{ijk} and z_{ik} that satisfy all the constraints and such that:

$$d_{ijk}^{(t)} y_{ijk} = 0 \quad \text{and} \quad h_{ik}^{(t)} z_{ik} = 0$$

This means that y_{ijk} can take the value 1 only if $d_{ijk}^{(t)} = 0$ and that z_{ik} can take the value 1 only if $h_{ik}^{(t)} = 0$.

The existence of a complete ordering of the jobs once all cycle remover constraints have been added will now be considered in the following theorem.

Theorem 7.5

A complete ordering of the jobs that is consistent with constraints (7.9),..,(7.19) exists if, and only if, no constraints with a positive multiplier can be introduced into the Lagrangean dual.

Proof

From graph theory, it is well known that a directed graph defines a partial ordering of the vertices if, and only if, it contains no cycles.

Here we form a directed graph with two classes of nodes. Jobs $1,\ldots,n$ form the first class of nodes and pairs of jobs ij, $i,j=1,\ldots,n$ and $i \neq j$ formthe second class of nodes (dummy nodes). An arc from node i to node j exists in this graph if $h_{ij}^{(t)} > 0$ while an arc from node ij to node k exists if $d_{ijk}^{(t)} > 0$.

If a cycle exists in this graph, then it is possible to remove this cycle by adding a constraint of type (7.19). If, on the other hand, no cycle exists, then it is possible to find an ordering of the jobs $(\pi'(1),\ldots,\pi'(n))$ which is consistent with the graph. The reverse ordering $(\pi'(n),\ldots,\pi'(1))$ is the required ordering of the jobs which is consistent with constraints (7.9),...,(7.19).

Once a complete ordering of the jobs is found, it becomes possible to evaluate this sequence. The value of this sequence forms an upper bound $UB^{(N+t)}$ on the value of the optimum. However, $UB^{(N+t)}$ can be found as follows:

$$UB^{(N+t)} = LB^{(N+t)} + \sum_{v=1}^{t} \lambda^{(t)} \left(\sum_{k=1}^{n} \sum_{i=1}^{n} g_{ik}^{(v)} z_{ik} + \sum_{k=1}^{n} \sum_{j=1}^{n} \sum_{i=1}^{n} e_{ijk}^{(v)} y_{ijk} - 1 \right) \tag{7.26}$$

It is clear from (7.26) that $UB^{(N+t)} = LB^{(N+t)}$ if, and only if, every one of the added t constraints of type (7.19) has one variable only (either one of the y_{ijk} or one of the z_{ik} variables) with a value equal to one. Thus we have the following theorem.

Theorem 7.6

$LB^{(N+t)} = UB^{(N+t)}$ if, and only if, all the constraints are satisfied as equalities.

From here to the end of this chapter, by lower bound we shall mean the lower bound as proposed in this section.

7.5 Implementation of the Lower Bound

The lower bounding procedure as described in the previous section requires a three dimensional array of size n x n x n to store the cost data. This made the storage requirements for the problem (even for n=40) out of the reach of computers. However, the size of the cost array is reduced as follows. From the previous section we have seen that y_{ijk} is equivalent to y_{jik} $(i \neq j \neq k \neq i)$ and that the cost of scheduling job k after job i, h_{ik}, is equal to $c_{iik} + c_{ikk} + c_{kik}$. With this in mind, it is possible to introduce new variables y'_{ijk} $(i,j,k=1,\dots,n$ and $i<j)$ defined as follows:

$$y'_{ijk} = \begin{cases} 1, & \text{if job k is not to be sequenced before jobs i and j.} \\ 0, & \text{otherwise.} \end{cases}$$

Thus problem (7.8) can be written as follows:

$$\text{Minimize} \quad \sum_{k=1}^{n} \sum_{j=2}^{n} \sum_{i=1}^{j-1} c'_{ijk}\, y'_{ijk} + \sum_{k=1}^{n} p_k^{\,2}\, w_k \tag{7.27}$$

Where c'_{ijk} is defined as follows:

$$c'_{ijk} = \begin{cases} 2p_i p_j w_k & \text{if } j \neq k \neq i \text{ and } i<j \\ 2p_i p_j w_k + p_j^{\,2}\, w_k & \text{if } k=i \text{ and } i<j \\ 2p_i p_j w_k + p_i^{\,2}\, w_k & \text{if } k=j \text{ and } i<j \end{cases} \tag{7.28}$$

Subject to constraints (7.9),...,(7.19), where each variable y_{ijk} $(i,j,k=1,\dots,n$ and $i \neq j \neq k \neq i)$ is replaced by y'_{ijk} if $i<j$ and by y'_{jik} if $i>j$. Also, each variable z_{ik} $(i,k=1,\dots,n$ and $i \neq k)$ is replaced by y'_{ikk} if $i<k$ and by y'_{kik} if $i>k$.

We remark that a new directed graph $G_{Y'}$ can be obtained from graph $G_{Z,Y}$ by coalescing equivalent nodes.

The cost array has now become of size $n^2(n-1)/2$. Every element of this array is involved only once when adding the equality constraints. Thus, we have $n(n-1)/2$ constraints involving two different jobs (since for every i and j such that $i<j$ (i.e. every row of the cost array) there exists two values for k such that $k=i$ or $k=j$ and because we have $n(n-1)/2$ rows). Also, we have $n(n-1)(n-2)/6$ constraints involving three different jobs (where $(n-2)$ is the number of different values that k can take (for each value of i and j, $i<j$) such that $k\neq i$ and $k\neq j$).

To store this cost array we used a two dimensional array $R(I,K)$, where $I=1,\ldots,n(n-1)/2$ and $K=1,\ldots,n$. Given i, j and k, the corresponding element of the cost arry is $R(I,k)$ where $I = i\times(n-1) - i\times(i-1)/2 - n+j$. *We shall refer to this method of storing the cost array as implementation 1.*

The array $R(I,K)$ was used to store the cost array for problems of sizes up to 40 jobs. Although it might be possible to store the cost array for problems of size 50 in two arrays (instead of one), we have saved half the storage requirements as follows. To store the cost array for problems of sizes larger than 40 (i.e. 50, 60 and 70) we used an array $R'(I,K')$ where I as above and $K'=1,\ldots,n/2$ (n is even). This was done by forming strings where every string $R'(I,K')$ consists of two elements of the cost array. Given two elements of the cost array, c'_{ijk} $(k \leq n/2)$ and $c'_{ij,k+n/2}$, a string is formed as follows: $c'_{ijk} + 10^7 \times c'_{ij,k+n/2}$, where 10^7 is larger than the maximum value of any element.

Thus, given i, j and k, the corresponding element of the cost array is given by $\lfloor R'(I,k-n/2)/10^7 \rfloor$ if $k > n/2$ and by $E - 10^7 \times \lfloor E/10^7 \rfloor$ if $k \leq n/2$, where $E = R'(I,k)$. Using these strings enabled us to solve problems of sizes up to 70 jobs. *We shall refer to this method of storing the cost array as implementation 2.*

We shall assume that one of the two implementation procedures discussed above is used.

As explained in Section 7.6, the bounding procedure is started by adding all possible equality constraints involving two different jobs (i.e. of type (7.9)). This requires $0(n^2)$ steps. We then add all possible equality constraints involving three different jobs. This requires $0(n^3)$ steps. We then see whether the reduced cost array defines a complete ordering of the jobs. This is done by spending $0(n^3)$ steps trying to schedule the jobs one by one from the end. We schedule a job k with $c'_{ijk} = 0$, for all unscheduled jobs i and j in the last available position. Column k and all rows involving job k are removed from the array. We repeat this until all jobs have been sequenced or until a stage where none of the jobs can be sequenced last is reached. If all jobs have been sequenced, we stop; our bounding procedure has ended. Since (according to Theorem 7.6) the lower bound computed in this case is equal to the upper bound obtained by evaluating the resulting sequence, the problem has been solved without the need for branching. If, on the other hand, a stage where none of the remaining jobs can be sequenced in the last available position is reached, we reduce the size of the problem by removing all the sequenced jobs. We then have the problem of adding all possible constraints of type (7.11), ...,(7.17) and of type (7.19) (if needed).

It is clear from Theorem (7.6) that it is desirable that any of the constraints we add should be satisfied as an equality. With this in mind we use a heuristic to indicate which constraints to add. This heuristic is to order the jobs in a non-increasing order of w_i/p_i. Let π be the sequence obtained using this heuristic. Initial experiments indicated that constraints (7.12) are satisfied as equalities more often than the others. Hence, we decided to add all possible constraints of type (7.12) and the condition that jobs i and j are sequenced before job k in

the sequence π. This procedure requires $0(n^3)$ steps. Obviously the possibility of these constraints being satisfied as equalities depends on how close the sequence π is to the optimum sequence.

Having added all possible constraints of type (7.12) we follow the procedure described above to see if the current reduced cost array defines a complete ordering of the jobs. If the answer is yes, then we have completed the lower bounding procedure. Otherwise, for each (i,j,k) $(i,j,k=1,\dots,n$ and $i<j)$ we add all possible constraints of type (7.13), (7.14), (7.15), (7.12), (7.16), (7.17) and (7.11) in that order. Applying (7.12) (for the second time) here might produce new constraints since k need not be sequenced after i and j in π. This procedure requires $0(n^3)$ steps.

Initial experiments indicated that the order in which we look for these constraints is of great significance.

Having added all possible constraints involving three jobs, we apply the procedure described above to sequence the jobs one by one from the end. Although a sequence (which satisfies all the constraints) for each of the 700 problems considered has been found at this stage of the lower bounding procedure, it is always possible to find problems for which further constraints need to be added. These constraints are of type (7.19). It is clear that these constraints are more likely to be satisfied as equalities if r is small. Hence it is wise to start by spending $0(n^4)$ steps adding all possible constraints involving four jobs (i.e. $r=4$). If a sequence cannot be found here, then constraint (7.19) is applied in its general form. This is done as follows. Consider a set of r different jobs $\{1,\dots,r\}$. Then it is possible to add the following constraint:

$$y'_{i_1j_1 1} + y'_{i_2j_2 2} + \dots + y'_{i_rj_r r} \geq 1$$

where i_v, $j_v \in \{1,\dots,r\}$, $i_v < j_v$ and that $c'_{i_vj_vv}$ is the largest element in column v $(v=1,\dots,r)$. Clearly, selecting the largest element in a

column requires $O(r^2)$ steps. Hence, adding a constraint involving r jobs requires $O(r^3)$ steps. In the worst case, up to $r(r-1)/2$ (number of rows) constraints may be found and thus the procedure requires $O(r^5)$.

We summarize, adding all possible equality constraints of type (7.9) and (7.10) requires $O(n^2)$ and $O(n^3)$ steps respectively. Adding all possible constraints of type (7.12) requires $O(n^3)$ steps. A further $O(n^3)$ steps is required when adding constraints of type (7.13), (7.14), (7.15), (7.12), (7.16), (7.17) and (7.11). Adding constraints of type (7.19) requires $O(r^5)$ in the worst case. Finally, the procedure to see whether the reduced cost array defines a complete ordering of the job requires $O(n^3)$ steps. As mentioned above, this procedure is applied after adding equality constraints (7.10) and after adding constraints of type (7.12) if such constraint were needed (i.e. if a complete ordering of the jobs was not obtained after adding constraints (7.9) and (7.10). If a complete ordering of the jobs cannot be found, then the procedure is repeated for the third time after adding all possible constraints involving three jobs. As mentioned above, constraints of type (7.19) were not needed for any of the 700 problems tested and that the reduced cost array, at this stage, defined a complete ordering of the jobs. Obviously, if constraints of type (7.19) were needed for a problem, then a further $O(n^3)$ steps is required to order the jobs.

7.6 Example

In this section, we shall explain our bounding procedure using an example given in (Townsend, 1978) (see Section 7.3). The initial cost array, $c = (c_{ijk})$ where $c_{ijk} = p_i p_j w_k$, for this example is:

Table 7.1: Initial Cost Array

i	j	k= 1	2	3	4	5
1	1	---	500	700	300	100
	2	80	200	280	120	40
	3	120	300	420	180	60
	4	20	50	70	30	10
	5	40	100	140	60	20
2	1	80	200	280	120	40
	2	32	---	112	48	16
	3	48	120	168	72	24
	4	8	20	28	12	4
	5	16	40	56	24	8
3	1	120	300	420	180	60
	2	48	120	168	72	24
	3	72	180	---	108	36
	4	12	30	42	18	6
	5	24	60	84	36	12
4	1	20	50	70	30	10
	2	8	20	28	12	4
	3	12	30	42	18	6
	4	2	5	7	---	1
	5	4	10	14	6	2
5	1	40	100	140	60	20
	2	16	40	56	24	8
	3	24	60	84	36	12
	4	4	10	14	6	2
	5	8	20	28	12	---

As mentioned in Section 7.5, it is possible to reduce the size of this cost array to form a new cost array $c' = (c'_{ijk})$ (where c'_{ijk} is given by (7.28)):

Table 7.2: Initial Cost Array (Reduced Size)

i	j	k= 1	2	3	4	5
1	2	192	900	560	240	80
	3	312	600	1540	360	120
	4	42	100	140	360	20
	5	88	200	280	120	140
2	3	96	420	448	144	48
	4	16	45	56	72	8
	5	32	100	112	48	32
3	4	24	60	91	144	12
	5	48	120	196	72	60
4	5	8	20	28	24	5

It is possible to add the following equality constraints:

$y'_{121} + y'_{122} = 1$ with $2\mu_{12} = 192$

$y'_{131} + y'_{133} = 1$ " $2\mu_{13} = 312$

$y'_{141} + y'_{144} = 1$ " $2\mu_{14} = 42$

$y'_{151} + y'_{155} = 1$ " $2\mu_{15} = 88$

$y'_{232} + y'_{233} = 1$ " $2\mu_{23} = 420$

$y'_{242} + y'_{244} = 1$ " $2\mu_{24} = 45$

$y'_{252} + y'_{255} = 1$ " $2\mu_{25} = 32$

$y'_{343} + y'_{344} = 1$ " $2\mu_{34} = 91$

$y'_{353} + y'_{355} = 1$ " $2\mu_{35} = 60$

$y'_{454} + y'_{455} = 1$ " $2\mu_{45} = 5$

$y'_{123} + y'_{132} + y'_{231} = 1$ with $\gamma = 96$

$y'_{124} + y'_{142} + y'_{241} = 1$ " $\gamma = 16$

$y'_{125} + y'_{152} + y'_{251} = 1$ " $\gamma = 32$

$y'_{134} + y'_{143} + y'_{341} = 1$ " $\gamma = 24$

$y'_{135} + y'_{153} + y'_{351} = 1$ " $\gamma = 48$

$y'_{145} + y'_{154} + y'_{451} = 1$ " $\gamma = 8$

$y'_{234} + y'_{243} + y'_{342} = 1$ " $\gamma = 56$

$y'_{235} + y'_{253} + y'_{352} = 1$ " $\gamma = 48$

$y'_{245} + y'_{254} + y'_{452} = 1$ " $\gamma = 8$

$y'_{345} + y'_{354} + y'_{453} = 1$ " $\gamma = 12$

Total contribution to Lower Bound 1635

The cost array becomes that given in Table 7.3.

Table 7.3: Reduced Cost Array

i	j	k= 1	2	3	4	5
1	2	0	708	464	224	48
	3	0	504	1228	336	72
	4	0	84	116	318	12
	5	0	168	232	112	52
2	3	0	0	28	88	0
	4	0	0	0	27	0
	5	0	68	64	40	0
3	4	0	4	0	51	0
	5	0	72	136	60	0
4	5	0	12	16	19	0

We next see if the reduced cost array defines a complete ordering of the jobs. It is clear that job 1 can be sequenced last since $c'_{ij1} = 0$ for all jobs i and j. We then remove column 1 and every row involving job 1. For the same reason we can sequence job 5 directly before job 1. Column 5 and all rows involving job 5 are removed from the cost array. No other jobs can be sequenced next. Removing columns 1 and 5 and all rows involving jobs 1 and 5, the cost array becomes:

Table 7.4: Reduced Cost and Size Array

i	j k=	2	3	4
2	3	0	28	88
	4	0	0	27
3	4	4	0	51

According to Theorem 7.5, it is possible to add at least one constraint. Since $c'_{342} > 0$, $c'_{233} > 0$ and $c'_{244} > 0$, it is possible to add the following constraint (of type 7.12):

$$y'_{342} + y'_{233} + y'_{244} \geqslant 1$$

with contribution to the lower bound equal to 4.

It is clear that all jobs can be sequenced. A unique sequence (4,3,2,5,1) can be found which is defined by values y'_{ijk} satisfying all constraints and $c'_{ijk}\, y'_{ijk} = 0$. The lower bound is given by:

$$LB = 1635 + 4 + \sum_{k\ 1}^{n} p_k^{\ 2}\, w_k$$

$$= 1635 + 4 + 539$$

$$= 2178$$

With regard to the last constraint added, one can see that it is satisfied as equality constraint (since only $y'_{342} = 1$) and consequently the upper bound obtained by evaluating this sequence is equal to the lower bound. Thus the problem has been solved without the need to perform any branchings.

7.7 The Algorithm

In this section we give a complete description of the algorithm, apart from the lower bounding procedure which has already been described in the previous sections. Here we shall be interested in describing the heuristic used, the branching rule and the search strategy.

The algorithm starts by applying the first heuristic which sequences the jobs in a non-increasing order of w_i/p_i. Although the value of this sequence is used as an upper bound on the optimum, the main purpose of this heuristic is to indicate which constraints are to be added when computing the lower bound.

We then apply the bounding procedure as described in Sections 7.4 and 7.5. The sequence π' obtained at the end of the bounding procedure is evaluated and the upper bound is updated accordingly. If the lower bound obtained is equal to the upper bound, the problem is solved without the need for branching. If, on the other hand, the lower bound is less than the upper bound, our second heuristic is applied which tries to improve the best sequence obtained. This is done as follows. Let $\pi = \{\pi(1),\pi(2),\ldots,\pi(n)\}$ be the sequence with the best solution obtained so far. Job $\pi(1)$ is removed from its position and temporarily sequenced in the second position, (i.e. after $\pi(2)$). If the resulting sequence is better than the original one, then job $\pi(1)$ is sequencedin its temporary position; otherwise, job $\pi(1)$ is considered for the third, fourth,...,nth position in a similar way. As soon as an improvement is made, this job is left in its

temporary position. If, on the other hand, no improvement can be made, job $\pi(1)$ is replaced in its original position.

This procedure is applied for all jobs $\pi(i)$, $i=2,\ldots,n$. Whenever an improvement in the value of the solution is made, the procedure is repeated from the beginning (i.e. from job $\pi(1)$). The procedure ends when no improvement can be made. This heuristic requires $0(n^2)$ steps if π is optimum. No such bound exists in the worst case.

The branching rule used here is similar to that used by Potts (Potts, 1981). The notations of Section 7.5 will be used to describe the branching rule. The idea behind the branching rule is to reduce the difference between the lower and upper bounds calculated at the node from which we are about to branch (by upper bound here we mean the value of the sequence π' obtained at the end of the lower bounding procedure).

A constraint which is satisfied as inequality and has a multiplier as large as possible is selected. One of the variables involving two jobs i and j only, i.e. y'_{ijj}, $i<j$ (or y'_{jij}, $i>j$) which occurs in this constraint is chosen for which $y'_{ijj} = 1$ ($y'_{jij} = 1$) and that no arc between job i and job j in the transitive closure of the precedence graph formed by considering all previous branchings (if any). Two branches of the search tree can then be formed, namely job i is constrained to be sequenced before job j and job i is constrained to be sequenced after job j. When job i is constrained to be sequenced before job j, each constraint involving the variable y'_{ijj} (y'_{jij}) is removed from the Lagrangean, without altering the lower bound, since it will necessarily be satisfied. We then update the transitive closure of the precedence graph (this part of the algorithm is not performed in the first level of the search tree). Whenever a new arc (h,k) for which $y'_{hkk} = 1$, $h<k$ (or $y'_{khk} = 1$ if $h>k$), all constraints involving y'_{hkk} (y'_{khk}) are removed from the Lagrangean. Given two jobs u and v, we also remove all constraints involving y'_{uvk} if $y'_{ukk} = 1$ ($y'_{kuk} = 1$) and $y'_{vkk} = 1$ ($y'_{kvk} = 1$).

The second case where job i is constrained to be sequenced after job j is dealt with in a similar way. It is clear that in this case where $y'_{ijj} = 0$ ($y'_{jij} = 0$) the sequence π' will be infeasible and hence a new sequence will have to be found.

Finally, our search strategy is given. A newest active node search is used which selects a node from which to branch which has $i<j$ and job i is constrained to be sequenced before job j.

We point out that all constraints of type (7.11),...,(7.17) are stored (constraints (7.19) were not needed). Each of these constraints is identified according to the level of the search tree and whether this constraint was added to or removed from the Lagrangean problem. Hence, whenever backtracking is necessary, the reduced cost arry, for a particular node at level h of the search tree, is recomputed by considering only those constraints that were added or removed at or after level h.

7.8 Incorporating the dominance Rules with the Lower Bound

Although we have not needed to use the dominance rules given in Section 7.2, it is possible to incorporate Theorem 7.1 and Corollary 7.1 with the lower bound. This is done (in the notation of Section 7.5) as follows. If job i dominates job j, then the value of c'_{ijj}, $i<j$ (c'_{jij} if $i>j$) is added to the lower bound. For $h=1,\ldots,n$, ($h\neq j$), the value of c'_{hji}, $h<j$ (c'_{jhi} if $h>j$) is increased to take a very large value M (e.g. the sum of all elements). Also, if each of two jobs i and j dominates a third job k, then the value of c'_{ijk}, $i<j$ (c'_{jik} if $i>j$) is added to the lower bound. For $h=1,\ldots,n$ ($h\neq k$), the values of c'_{hki} and c'_{hkj} if $h<k$ (c'_{khi} and c'_{khj} if $h>k$) are increased to take very large values of M.

With regard to the example in Section 7.6, we have the following. From Theorem 7.1 and Corollary 7.1, jobs 2, 3 and 4 must be sequenced before jobs 1 and 5. Thus it is possible to add the values of the following

elements of the cost array to the lower boud: c'_{121}, c'_{131}, c'_{141}, c'_{255}, c'_{355}, c'_{455}, c'_{231}, c'_{241}, c'_{341}, c'_{235}, c'_{245} and c'_{345} (i.e. a total of 847 is added to the lower bound). For $h=1,\ldots,5$, the values of c'_{1h2}, c'_{1h3}, c'_{1h4} (where $h\neq 1$), and c'_{h52}, c'_{h53}, c'_{h54} (where $h\neq 5$) are set to equal M. Thus the initial cost array of Table 7.2 becomes as given in Table 7.5.

Table 7.5

i	j	k= 1	2	3	4	5
1	2	0	M	M	M	80
	3	0	M	M	M	120
	4	0	M	M	M	20
	5	88	M	M	M	140
2	3	0	420	448	144	0
	4	0	45	56	72	0
	5	32	M	M	M	0
3	4	0	60	91	144	0
	5	48	M	M	M	0
4	5	8	M	M	M	0

7.9 Precedence Constraints

The proposed lower bounding procedure can also be used for the more general case where precedence constraints among jobs are specified.

Given a directed graph G representing precedence constraints, $A = (a_{ij})$ is the adjacency matrix of G where $a_{ij} = 1$ if an arc (i,j) exists in the transitive closure of G and $a_{ij} = 0$, otherwise.

Besides constraints (7.5), (7.6) and (7.7) of Section 7.4, we have the following constraint:

$$x_{ij} \geq a_{ij}, \qquad i,j=1,\ldots,n.$$

Using the notation of Section 7.5, this constraint can be written as:

$$y'_{ijj} \geq a_{ij}, \qquad i,j=1,\ldots,n \text{ and } i<j$$

or

$$y'_{jij} \geq a_{ij}, \qquad i,j=1,\ldots,n \text{ and } i>j$$

Thus, if $a_{ij} = 1$ $(i<j)$, the value of the element c'_{ijj} can be added to the lower bound. For $h=1,\ldots,n$ $(h \neq j)$, the elements c'_{hji} if $h<j$ (c'_{jhi} if $h>j$) are increased to a very large number M.

It is obvious that if $a_{ik} = 1$ and $a_{jk} = 1$, then $y'_{ijk} = 1$, which means that we can add the value of the element c'_{ijk} to the lower bound. Also, for $h=1,\ldots,n$ $(h \neq k)$, we have: $c'_{hki} = c'_{hkj} = M$ for $h<k$ ($c'_{khi} = c'_{khj} = M$ for $h>k$).

With regard to the example of Section 7.6, suppose that we are given the precedence graph G (Figure 7.3).

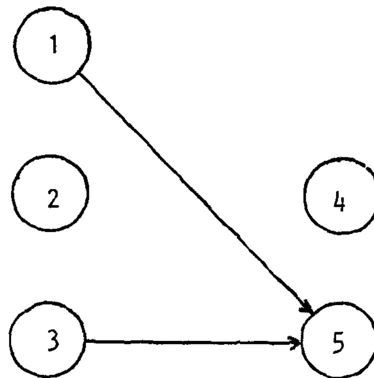


Figure 7.3: Precedence Graph G

It is clear that $a_{15} = 1$ and $a_{35} = 1$ imply that $y'_{155} = y'_{355} = y'_{135} = 1$. Hence, we can add the values of the elements c'_{155}, c'_{355} and c'_{135} to the lower bound (a total of 320). Also, for $h=1,\ldots,4$, set $c_{h51} = c_{h53} = M$. Thus the initial cost array of Table 7.2 has now become that given in Table 7.6.

Table 7.6

i	j	k= 1	2	3	4	5
1	2	192	900	560	240	80
	3	312	600	1540	360	0
	4	42	100	140	360	120
	5	M	200	M	120	0
2	3	96	420	448	144	48
	4	16	45	56	72	8
	5	M	100	M	48	32
3	4	24	60	91	144	17
	5	M	120	M	72	0
4	5	M	20	M	24	5

To obtain a lower bound for this problem, we suggest the following. Add all possible constraints of type (7.9) and (7.10). Then use a heuristic to obtain a sequence π consistent with the precedence constraints (see Chapter 5 for heuristics which can be adapted for this problem). Use the sequence π to indicate which constraints of type (7.12) are to be added. Sequence π may also be used to indicate which constraints of type (7.13), ...,(7.17) are to be added. The order in which to look for these constraints may need to be changed. Then all possible constraints of type (7.19) are added as described in Section 7.5, after which a complete ordering of the jobs exists.

It may be wise, at this stage, to improve the bound by taking further steps similar to the procedure taken by Potts (Potts, 1981) to improve his bound for the single machine sequencing with precedence constraints. The procedure can be summarized as follows.

Let $\pi' = (\pi'(1),\ldots,\pi'(n))$ be the sequence obtained at the end of the above bounding procedure. Suppose also that a constraint satisfied as inequality with multipler λ has been found. Suppose that the variable y'_{ijk} occurs in this constraint, where jobs i and j are not sequenced after k in π', which implies that $y'_{ijk} = 1$. There are at least two such variables

in each constraint satisfied as an inequality. This constraint is temporarily removed from the Lagrangean by increasing the appropriate elements of the cost array by λ. Suppose this leads to the existence of at least two constraints involving the variables y'_{ijk} in that constraint with multipliers summing up to λ'. If $\lambda' > \lambda$, then the original constraint is removed and the new constraints are added, this leads to increasing the lower bound by $\lambda'-\lambda$. In this case, finding a new ordering of the jobs may be necessary, since some of the elements of the cost array may have been increased from their zero values. If, on the other hand, $\lambda' \leq \lambda$ or it was not possible to find two new constraints, then the original constraint is reintroduced, the new constraints are ignored and the lower bound remains the same.

We should point out that this procedure to improve the lower bound can be used for the unconstrained case also.

7.10 Computational Experience

7.10.1 Test Problems

Every problem consists of n jobs, two integers were generated for every job i, namely p_i and w_i. Processing times p_i $(i=1,\ldots,n)$ were generated randomly from a uniform distribution [1,100]. Weights w_i $(i=1,\ldots,n)$ were generated from a uniform distribution [1,10].

A hundred problems were generated for every value of $n(n=10,20,\ldots,70)$; 700 problems in all were tested.

7.10.2 Computational Results

The algorithms were coded in FORTRAN IV and run on a CDC 7600 computer.

Initial experiments involving problems of size 20 and 30, showed our branch and bound procedure to dominate the one proposed by Townsend. For this reason we have excluded the results for Townsend's algorithm.

Results for our proposed branch and bound algorithm are given in Table 7.8. Minimum, average and maximum total numbers of added constraints of type (7.11),...,(7.17) are given in columns 1 to 3 respectively. Minimum, average and maximum numbers of nodes are given in the next three columns, followed by the average execution times in the last column.

The first three columns show that, as expected, the number of constraints needed increases as the size of the problem increases. Columns 4, 5 and 6 show that all problems but one have been solved without the need for branchings. This one case where branchings were needed occurred when n=60; even in this case the problem was solved in two nodes only. It is clear from column 7 that there was a big increase in the average execution time for problems of size 50 from what it was for problems of size 40. This jump occurred because of the fact that, as explained in Section 7.5, strings were introduced to enable the computer to solve problems of size 50 or larger.

A closer look at column 2 shows that as the number of jobs increases from 10 to 20, average number of added constraints increases by about 650%. This rate of increase in the average number of added constraints decreases as the number of jobs increases and reaches its minimum value of 60% when the number of jobs is increased from 50 to 60 and from 60 to 70 (equal rate of increase). A closer look at column 7 shows that computation time increases by 622% as the number of jobs increases from 10 to 20. This rate of increase in computation time decreases as the number of jobs increases and takes the value 142% as n increases from 30 to 40. Using implementation 2 (i.e. forming strings) lead to an increase of about 286% in computation time as n increases from 40 to 50. This sharp increase in computation time (compared with 78% increase in average number of added constraints) decreases as n increases and reaches its minimum value of about 59% as n increases from 60 to 70.

Table 7.8*

	n	1	2	3	4	5	6	7
				Column Number**				
Implementation 1	10	0	6	26	0	0	0	.00
	20	4	43	159	0	0	0	.03
	30	23	119	284	0	0	0	.12
	40	127	255	450	0	0	0	.29
Implementation 2	50	211	454	956	0	0	0	1.12
	60	410	728	1238	0	0.02	2	2.34
	70	738	1165	2227	0	0	0	3.71

*Times are in CPU seconds.

**Column Number:

1. Minimum number of added constraints of type (7.11), ...,(7.17).
2. Average number of added constraints of type (7.11), ...,(7.17).
3. Maximum number of added constraints of type (7.11), ...,(7.17).
4. Minimum number of nodes.
5. Average number of nodes.
6. Maximum number of nodes.
7. Average computation times.

7.11 Concluding Remarks

In Section 7.2 we showed that problems with agreeable weights (i.e. $p_i \leqslant p_j \to w_i \geqslant w_j$) can be solved by ordering the jobs according to non-increasing order of w_i. We then, in Sections 7.4, 7.5 and 7.6, proposed a branch and bound procedure for solving the general case. As mentioned before, this branch and bound procedure was tested using randomly generated data. The excellent results we had were not expected. The order in which we looked for the constraints was a major factor behind these excellent results.

Initial experiments on problems of size 20 and 30 jobs with weakly correlated data: $11 \leq w_i \leq 100$ and $w_i - 10 \leq p_i \leq w_i + 10$, showed these problems to be unexpectedly harder than the randomly generated ones. In most cases, constraints of type (7.19) with $r \geq 4$, were needed. This caused the gap between the initial lower and upper bounds to be quite large for these problems. One way to improve these results might be by trying different orders in which we look for constraints of type (7.11), ...,(7.17).

PART III

MULTI-MACHINE SCHEDULING

CHAPTER EIGHT

FLOW-SHOP SCHEDULING

8.1 Introduction

The general flow-shop problem, indicated by $F//C_{max}$, can be stated as follows. There are n jobs numbered 1,...,n, each of which is to be processed on machines 1,...,m in that order. Each job i (i=1,...,n) has a processing time p_{ik} on machine k (k=1,...,m). Each machine can process not more than one job at a time and each job can be processed by not more than one machine at a time. Once the processing of a job on a machine has started, it must be completed without interruption. The order in which jobs are processed need not be the same on all machines. The objective is to find a processing order on each machine which minimizes C_{max}, the maximum completion time of all the jobs.

It is well known (Conway et al., 1967; Rinnooy Kan, 1976; Lenstra, 1977) that to find the optimum for the $Fm//C_{max}$ problem, we need to consider only schedules with the same processing order on the first two machines and the same processing order on the last two machines. The following two-job $F4//C_{max}$ example, given in (Conway et al., 1967), shows that this result cannot be extended any further: Let $p_{11} = p_{22} = p_{23} = p_{14} = 4$, $p_{21} = p_{12} = p_{13} = p_{24} = 1$. There are only two order-preserving schedules (see Figure 8.1), both of which have a maximum completion time of 14.

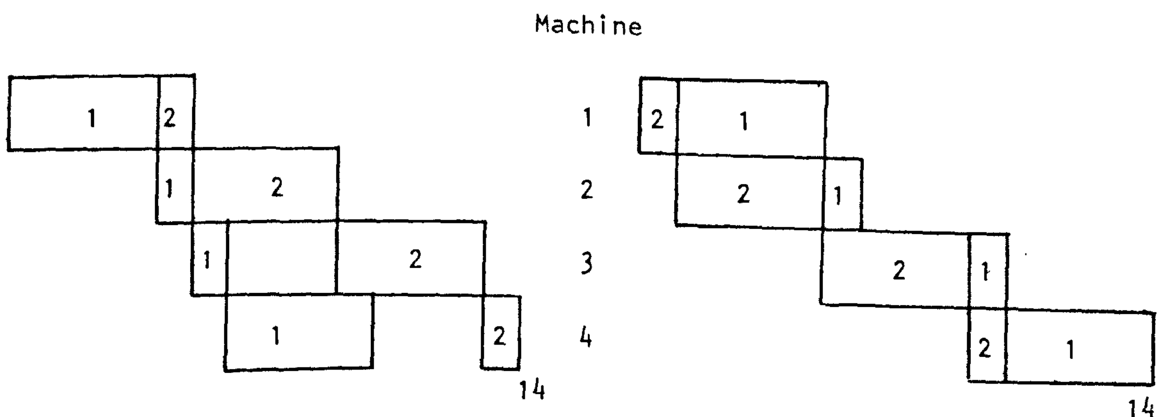


Figure 8.1

Now consider a schedule (see Figure 8.2) which has the same order on machines 1 and 2 and the same processing order on machines 3 and 4, but in which the order is reversed between machines 2 and 3. The maximum completion time is 12, which is less than what was obtained above.

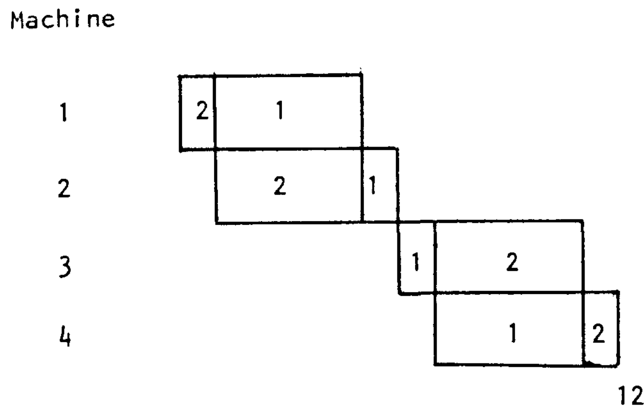


Figure 8.2

If each job has an imposed sequence of operations which may differ from the sequence of operations of other jobs, the problem is known as the *job-shop* problem and is denoted by $Jm//C_{max}$. If the sequence of operations for each job is not imposed but is to be chosen by the scheduler, the problem is known as the *open-shop* problem and is denoted by $Om//C_{max}$. If, on the other hand, we restrict ourselves to minimization over all schedules with the same order on each machine, the resulting problem is called the *permutation flow-shop* problem which is denoted by $Pm//C_{max}$.

The above result regarding the processing order on the first two and last two machines for the $Fm//C_{max}$ problem implies that the $F2//C_{max}$ and $P2//C_{max}$, and the $F3//C_{max}$ and $P3//C_{max}$ problems are equivalent.

Finally, it is also well known that for m=2, the resulting flow-shop problem, i.e. $F2//C_{max}$, can be solved using Johnson's algorithm (Johnson, 1954) in which job i is sequenced before job j if $\min(p_{i1},p_{j2}) \leq \min(p_{i2},p_{j1})$. This algorithm requires $O(n \log n)$ steps. If precedence constraints in the form of *Series-Parallel* graph $G=(V,E)$ (where an arc

$(i,j)\epsilon E$ implies that job i must be processed before job j on each machine) were added to the problem, the resulting problem can still be solved using Sidney's algorithm (Sidney, 1979). However, for general precedence constraints, the $F2/prec/C_{max}$ problem is NP-hard (Monma,). For full details about this general problem, we refer to Chapter 9. The $F2/r_i/C_{max}$ and $F3//C_{max}$ problems have been shown to be NP-hard also (Garey, Johnson & Sethi, 1976; Lenstra, Rinnooy Kan & Brucker, 1977).

Remark

As we pointed out, C_{max} is the optimality criteria to be used in this chapter. Using other criteria usually lead to NP-hard problems. The following problems have already been shown to be NP-hard.

$F2//L_{max}$	(Lenstra et al., 1977)
$F2//\Sigma C_i$	(Garey et al., 1976)
$O2//L_{max}$	(Lawler et al., 1981)
$Om//\Sigma C_i$	(Gonzalez, 1979)
$F2/pmtn/L_{max}$	(Cho & Sahni, 1978)
$F3/pmtn/\Sigma C_i$	(Lenstra, 1981)
$J2/pmtn/\Sigma C_i$	(Lenstra, 1981)
$Om/pmtn/\Sigma C_i$	(Gonzalez, 1979)

Only the $Om/pmtn,r_i/L_{max}$ problem can be solved in polynomial time by using linear programming (Cho & Sahni, 1978). Two other problems: $O2//\Sigma C_i$ and $F2/pmtn/\Sigma C_i$ are still open (Lawler, Lenstra & Rinnooy Kan, 1981).

In this chapter, we shall mainly concentrate on the permutation flow-shop problem, branch and bound algorithms for which will be reviewed in Section 8.2. A brief discussion of the open- and job-shop problems will be given in Section 8.3.

8.2 The $Pm//C_{max}$ Problem

8.2.1 Branching Rule

Most published algorithms for the permutation flow-shop problem (see Ignall & Schrage, 1965; Lomnicki, 1965; Brown & Lomnicki, 1966; McMahon & Burton, 1967; Nabeshima, 1967; Potts, 1974; Bestwick & Hastings, 1976; Lageweg, Lenstra & Rinnooy Kan, 1978) used the same branching rule in which each node of the search tree corresponds to a job being sequenced at the beginning. Hence, nodes at level h of the search tree correspond to initial partial sequences, each of which contains h fixed jobs. However, it is reported in (Potts, 1980A) that from computational results both Brown and Lomnicki (Brown & Lomnicki, 1966) and McMahon and Burton (McMahon & Burton, 1967) found that in some circumstances it is more efficient to solve the inverse problem which is obtained by interchanging the processing times p_{ik} and $p_{i,m-k+1}$ for all jobs i $(i=1,\ldots,n)$ and all machines k such that $1 \leq k \leq m/2$ rather than solving the original problem. This resulting problem, i.e. the inverse problem, is equivalent to a branching procedure for the original problem in which each node of the search tree corresponds to a job being sequenced at the end. Hence, nodes at level h of the search tree, in this case, correspond to final partial sequences, each of which contains h fixed jobs.

With this in mind Potts (Potts, 1980A) proposed an effective branching procedure. He called it the *adaptive branching rule*. Here, each node of the search tree corresponds to an initial partial sequence σ_1 and a final partial sequence σ_2, where either σ_1 or σ_2 may be empty. It is clear that Potts' branching rule reduces to the above given branching rule if σ_2 is empty.

Now, we shall give a full description of this adaptive branching rule. The first branching sequences a job in position 1 while the second branching sequences a job in position n. The following branchings will

either be of type 1 in which a job is added to the end of an initial partial sequence σ_1 or of type 2 in which a job is added to the beginning of a final partial sequence σ_2. Deciding between type 1 and type 2 branchings is done using the following rule. Let k_1 and k_2 denote the lowest levels of the search tree at which nodes were constructed from type 1 and type 2 respectively. Also let n_1 and n_2 be the numbers of nodes which have lower bounds equal to the minimum value bound at levels k_1 and k_2 respectively. Also let n_1 and n_2 be the numbers of nodes which have lower bounds equal to the minimum value bound at levels k_1 and k_2 respectively. If $n_1 < n_2$, the next branching is of type 1, while if $n_1 > n_2$, the next branching is of type 2. If $n_1 = n_2$, then the next branching will be the same as the previous one. If, at some level of the search tree, all nodes were eliminated by dominance or upper bounds, all the next branchings will be of the same type as the previous branching.

From computational results, Potts (Potts, 1980A) found that there are substantial savings in computation when using the adaptive branching rule than when using the usual one.

8.2.2 Lower Bounds

As mentioned before, the branch and bound techniques were first applied to scheduling problems by (Ignall & Schrage, 1965; Lomnicki, 1965; Brown & Lomnicki, 1966; McMahon & Burton, 1967).

The so-called *machine based bound* was used for the first time by Ignall and Schrage (Ignall & Schrage, 1965). Given an initial partial sequence σ with $C(\sigma,u)$ as the minimum completion time of jobs sequenced in σ on machine u and a set of unsequenced jobs S, the machine-based bound takes the following form:

$$\max_{u=1,\dots,m} \{r_{*u} + \sum_{i\epsilon S} p_{iu} + q_{*u}\} \qquad (8.1)$$

where $r_{*u} = \min_{i \in S}\{r_{iu}\}$

$$q_{*u} = \min_{i \in S}\{q_{iu}\}$$

$$r_{iu} = \max_{j=1,\ldots,u} \{C(\sigma,j) + \sum_{k=j}^{u-1} p_{ik}\}$$

and $q_{iu} = \sum_{j=u+1}^{m} p_{ij}$

Using r_{*u} instead of $C(\sigma,u)$ in 8.1 makes this bound slightly stronger than the bounds used in (Lomnicki, 1965; Brown & Lomnicki, 1966; McMahon & Burton, 1967).

The so-called *job-based bound* was used for the first time by McMahon & Burton (McMahon & Burton, 1967). "This new bound expresses the fact that the makespan (C_{max}) may be determined by the total processing time for a job, rather than by the total processing time on one machine." (McMahon & Burton, 1967). The job-based bound takes the following form:

$$\max_{u=1,\ldots,m-1} \{C(\sigma,u) + \max_{i \in S} \{ \sum_{k=u}^{m} p_{ik} + \sum_{h \in S-\{i\}} \min\{p_{hu}, p_{hm}\}\}\} \qquad (8.2)$$

Replacing $C(\sigma,u)$ by r_{*u} leads to a slightly stronger bound which was used by McMahon (McMahon, 1971).

Using two-machine subproblems (instead of one) to obtain lower bounds was developed independently by Potts (Potts, 1974) and Lageweg et al. (Lageweg et al., 1978). Johnson's $P2//C_{max}$ algorithm was used to solve each of the resulting two-machine subproblems.

This two-machine bound was generalized by Potts (Potts, 1980A) to give a lower bound on all completion time for all possible schedules starting with the initial partial sequence σ_1 and ending with the final partial sequence σ_2. In this section we shall give a full description of this generalized bound. We shall also show the relation between this bound and the previously published ones.

Firstly, some notation is introduced. Let S_1 and S_2 be the set of jobs sequenced in σ_1 and σ_2 respectively. Also, let S be the set of unsequenced jobs. We define $C_1(\sigma_1,j)$ to be the minimum completion time of all jobs sequenced in σ_1 on machine j and $C_2(\sigma_2,j)$ to be the minimum time between the start of processing jobs in σ_2 on machine j and the completion of processing jobs in σ_2 on machine m. (If $S_1=\emptyset$, we define $C_1(\sigma_1,j) = 0$, for all j also if $S_2=\emptyset$, we define $C_2(\sigma_2,j) = 0$).

Here, a lower bound is obtained by relaxing the capacity constraints on some machines, i.e. by allowing some of the machines to process more than one job at the same time. This is done by choosing a machine pair (u,v), where $1 \leq u \leq v \leq m$, and relaxing the constraint that machines $u+1,\ldots,v-1$ can process only one job at a time. If $u \neq v$, a two-machine subproblem results in which each job $i \in S$ has a processing time p_{iu} on the first machine, a processing time p_{iv} on the second machine and a time lag of $\sum_{k=u+1}^{v-1} p_{ik}$ between the completion of processing job i on machine u and the start of processing job i on machine v. This resulting subproblem can be solved by ordering the jobs using Johnson's rule for a two machine problem with processing times $\sum_{k=u}^{v-1} p_{ik}$ and $\sum_{k=u+1}^{v} p_{ik}$, $i \in S$ (Conway et al., 1967). On the other hand, if $u=v$, a single machine problem results for which any sequence is optimum. A lower bound for the problem is given by:

$$B(\sigma_1,\sigma_2,u,v) = r_{*u} + T(\sigma_1,\sigma_2,u,v) + q_{*v} \qquad (8.3)$$

where $T(\sigma_1,\sigma_2,u,v)$ denotes the minimum value of the maximum completion time for the subproblem,

$$r_{*u} = \min_{i \in S}\{r_{iu}\}$$

and $$q_{*v} = \min_{i \in S}\{q_{iv}\}$$

where:

$$r_{iu} = \begin{cases} C_1(\sigma_1, u), & \text{if } S_1 \neq \emptyset \\ \sum_{k=1}^{u-1} p_{ik}, & \text{if } S_1 = \emptyset \end{cases} \tag{8.4}$$

and

$$q_{iv} = \begin{cases} C_2(\sigma_2 v), & \text{if } S_2 \neq \emptyset \\ \sum_{k=v+1}^{m} p_{ik}, & \text{if } S_2 = \emptyset \end{cases} \tag{8.5}$$

A slightly stronger version of (8.4) and (8.5) can be written as follows:

$$r_{iu} = \max_{j=1,\ldots,u} \quad C_1(\sigma_1, j) + \sum_{k=j}^{u-1} p_{ik}$$

and

$$q_{iv} = \max_{j=v,\ldots,m} \quad C_2(\sigma_2, j) + \sum_{k=v+1}^{j} p_{ik}$$

Thus, an overall lower bound for the problem $LB(\sigma_1, \sigma_2, W)$ is given by specifying a set of machine pairs.

$$W = \{(u_1, v_1), \ldots, (u_w, v_w)\}$$

and hence

$$LB(\sigma_1, \sigma_2, W) = \max\{B(\sigma_1, \sigma_2, u_1, v_1), \ldots, B(\sigma_1, \sigma_2, u_w, v_w)\} \tag{8.6}$$

The lower bound $B(\sigma_1, \sigma_2, u, v)$ is a generalization of the lower bound used in (Lageweg et al., 1978) and (Potts, 1974) indicated by $B(\sigma_1, \phi, u, v)$. It is also a generalization of Nabeshima's lower bound (Nabeshima, 1967) defined by $B(\sigma_1, \sigma_2, u, u+1)$, $u=1,\ldots,m-1$. Finally, when $W = \{(1,1), \ldots, (m,m)\}$, the resulting bound is known as the machine based bound.

We should point out at this stage that Potts' lower bound given by (8.3.) will be generalized to be used for a more general problem where precedence constraints amongst jobs exist. This problem will be considered in Chapter 10.

In (Lageweg et al., 1978) and (Potts, 1974) it was found that the sets of machine pairs $\{(1,m),\ldots,(m-1,m)\}$ and $\{(1,m),\ldots,(m,m)\}$ respectively gave good computational results. In his paper Potts (Potts, 1980A) proposed the following set of machine pairs:

$$W_0 = \{(1,1),\ldots,(m,m),(1,m),\ldots,(m-1,m)\}$$

"One factor likely to affect the efficiency of $B(\sigma_1,\sigma_2,u,v)$ is the total processing time on machines u and v. Larger total processing times are expected to produce higher bounds. Another factor is the size of v-u: the poor results obtained by Ashour and Quraushi (1969) for Nabeshima's bound indicate that $B(\sigma_1,\sigma_2,u,v)$ is likely to increase as v-u increases. With this in mind we suggest two other choices of sets of machine pairs. Firstly, we define $W_1=W_0 \cup \{(u,v)\}$ if machines u and v can be found such that $1 \leq u < v < m$ and the total processing time on each of machines u and v exceeds the total processing time on all other machines; otherwise $W_1=W_0$. Secondly, we define $W_2 = W_0 - \{(u,u),(u,m)\}$ if a machine u can be found such that $(m-1)/2 \leq u < m$ and the total processing time on machine u is less than the total processing on all other machines; otherwise $W_2=W_0$." (Potts, 1980A). However, W_0 appears to be computationally more effective than W_1 or W_2.

Computational results obtained by Potts indicated that the lower bound proposed by him, given by (8.3), is stronger than previously published ones. The results also showed that the set of machine pairs W_0 performed better than the set of machine pairs W_1 and W_2.

8.2.3 Dominance Rules

In this section we shall give some dominance rules under which a node can be eliminated before its lower bound is calculated. Clearly, these rules are particularly useful when a node with a lower bound that is less than the optimum, can be eliminated.

Let σ' and σ'' be two initial partial sequences and let S' and S'' be the sets of jobs sequenced in σ' and σ'' respectively. (Given a set S we define $\bar{S} = \{1,\ldots,n\}-S$.) We say that σ'' dominates σ' if for any permutation $\overline{\sigma'}$ of $\overline{S'}$ there exists a permutation $\overline{\sigma''}$ of $\overline{S''}$ such that $C_1(\sigma''\overline{\sigma''},m) \leq C_1(\sigma'\overline{\sigma'},m)$. We now have the following theorem.

Theorem 8.1 (Ignall & Schrage, 1965; Smith & Dudek, 1967; McMahon, 1969)

If $S' = S''$ and $C_1(\sigma'',k) \leq C_1(\sigma',k)$ for $k=1,\ldots,m$, then σ'' dominates σ'.

Theorem 8.1 is referred to as *the dynamic programming dominance theorem.* McMahon (McMahon, 1969) showed Theorem 8.1 to be the strongest possible one for the case when $S' = S''$: since if $C_1(\sigma'',k) > C_1(\sigma',k)$ for some machine k, then it is possible to choose the processing times for the unscheduled jobs (i.e. p_{ik} where $i\in\overline{S'}$) in such a way that $C_1(\sigma''\overline{\sigma''},m) > C_1(\sigma'\overline{\sigma'},m)$.

Several elimination criteria have been developed for the case $\sigma' = \sigma_1 j$ and $\sigma'' = \sigma_1 ij$ (i.e. $S'' = S'\cup\{i\}$). In the remainder of this section, we shall give conditions under which an initial partial sequence $\sigma_1 ij$ dominates another initial partial sequence $\sigma_1 j$, but first we have the following definition.

$$\Delta_{1k} = C_1(\sigma_1 ij,k) - C_1(\sigma_1 j,k), \quad (k=1,\ldots,m) \tag{8.7}$$

An initial partial sequence $\sigma_1 ij$ dominates an initial partial sequence $\sigma_1 j$ if one of the following conditions holds:

(i) (Smith & Dudek, 1969)

$$\Delta_{1k-1} \leqslant p_{ik} \quad \text{and}$$

$$C_1(\sigma_1 i,k-1) \leqslant C_1(\sigma_1 j,k-1), \quad (k=2,\ldots,m) \qquad (8.8.)$$

(ii) (McMahon, 1969; Szwarc, 1973)

$$\max\{\Delta_{1k-1},\Delta_{1k}\} \leqslant p_{ik}, \quad (k=2,\ldots,m) \qquad (8.9)$$

(iii) (Szwarc, 1971)

$$\Delta_{1k-1} \leqslant \Delta_{1k} \leqslant p_{ik} \quad (k=2,\ldots,m) \qquad (8.10)$$

(iv) (Szwarc, 1973)

$$\Delta_{1k} \leqslant \min_{u=k,\ldots,m} \{p_{iu}\} \quad (k=2,\ldots,m) \qquad (8.11)$$

(v) (Szwarc, 1973)

$$\max_{u=1,\ldots,m} \{\Delta_{1u}\} \leqslant p_{ik} \quad (k=2,\ldots,m) \qquad (8.12)$$

With respect to the above conditions, we have the following results.

<u>Theorem 8.2</u> (Szwarc, 1973)

Conditions (ii), (iii), (iv) and (v) are equivalent.

<u>Theorem 8.3</u> (Rinnooy Kan, 1976; Lenstra, 1977)

Condition (i) implies condition (ii).

Given a final partial sequence σ_2 and two unscheduled jobs i and j, we have the following definition:

$$\Delta_{2k} = C_2(ji\sigma_2,k) - C_2(j\sigma_2,k), \quad (k=1,\ldots,m) \qquad (8.13)$$

A final partial sequence $ji\sigma_2$ dominates $j\sigma_2$ if the following condition holds (Szwarc, 1971):

$$\Delta_{2k} \leq \Delta_{2k-1} \leq p_{ik-1}, \quad (k=2,\ldots,m) \tag{8.14}$$

It is obvious that condition (8.14) is symmetrical to condition (8.10) above.

"Computational experience reported in (McMahon, 1971; Baker, 1975) indicates that enumerative methods based on the simple elimination criteria above are inferior to those based on lower bounds; inclusion of these criteria in the latter type of algorithm leads to a gain in efficiency only for problems of moderate size ($n \leq 15$). Altogether, it seems that the elimination criteria discussed in this section are of little algorithmic value". (Lenstra, 1977).

However, dominance rules have been used as a part of branch and bound algorithms. Computational results obtained by Lageweg et al. (Lageweg et al., 1978) indicate that introducing dominance rules reduce computation. This result was confirmed in (Potts, 1980A).

8.2.4 Heuristic Methods

Dannenbring (Dannenbring, 1977) carried out some computational experiments to test the performance of several permutation flow-shop heuristics. In this section we shall talk about six of these heuristics.

The first of these heuristics is due to Palmer (Palmer, 1965) and is known as the *Slope Order heuristic* (SO). For each job i ($i=1,\ldots,n$), a slope index δ_i is calculated as follows:

$$\delta_i = \sum_{k=1}^{m} \left(k - \frac{m+1}{2}\right) p_{ik}$$

A sequence is then obtained by ordering the jobs according to non-incresing δ_i. The resulting sequence is then evaluated as a $Pm//C_{max}$ schedule. This procedure requires $O(\max\{mn, n \log n\})$ steps.

The second of these heuristics is due to Campbell, Dudek and Smith (Campbell et al., 1970) and will be referred to as the CDS heuristic. For each k(k=1,...,m-1), apply Johnson's (Johnson, 1954) algorithm for the $P2//C_{max}$ problem to solve a two-machine subproblem where each job i (i=1, ...,n) has processing times $\sum_{j=1}^{k} p_{ij}$ and $\sum_{j=m+1-k}^{m} p_{ij}$ on the first and second machines respectively. The resulting sequence is then evaluated as $Pm//C_{max}$ schedule. The best of the m-1 solutions is chosen as the heuristic solution to the m-machine problem. This procedure requires O(mn log n) steps.

The third heuristic will be referred to as the *Random sampling heuristic* (R) which selects solutions by randomly ordering the jobs.

The next three heuristics are due to Dannenbring (Dannenbring, 1977).

The first of these three heuristics is called the *Rapid Access procedure* (RA). This procedure is similar to that of Campbell, Dudek and Smith. Here, only a single two-machine subproblem is formed where each job i (i=1,...,n) has processing times $\sum_{j=1}^{m} (m-j+1)p_{ij}$ and $\sum_{j=1}^{m} (j)\, p_{ij}$ on the first and second machines respectively. Johnson's algorithm is used to solve this two-machine subproblem. The resulting sequence is then evaluated as $Pm//C_{max}$ schedule.

The second heuristic method is called the *Rapid Access with Close order Search* (RACS). Here, a simple interchange of each of the (n-1) pairs of adjacent jobs is examined for possible improvement in the objective function value.

The final heuristic is called the *Rapid Access with Extensive Search* (RAES). Instead of terminating the search after one set of interchanges, the RAES heuristic use the best immediate interchange to generate new interchanges. This procedure continues until no improvement in the value of the objective can be achieved.

It should be clear now that both heuristics SO and RA generate a single solution, while both heuristics CDC and R generate multiple solutions

from which the best is chosen. The CDS heuristic generates (m-1) solutions while the R heuristic generates as many or as few solutions as desired to be generated.

Dannenbring (Dannenbring, 1977) tested the performance of the above heuristics (among other heuristics) using 1580 problems ranging in size from 3 jobs, 3 machines to 50 jobs, 50 machines.

He used the branch and bound procedure given in (Dannenbring, 1973) to find optimum solutions to 1509 of the test problems. Estimates of the optimal solution value were obtained for the other 71 problems using the estimation procedure described in (Dannenbring, 1973).

Computational results in (Dannenbring, 1977) showed heuristic RAES to have the least percentage deviation from optimum and to be the most consistent of all the heuristics on the small sized problems (n x m: 3 x 3 to 6 x 10). The performance of heuristics RACS, CDS, R, RA, SO on the small sized problems follow in that order. With regard to problems of large sizes (n x m: 7 x 3 to 50 x 50), the results showed the RAES procedure to remain the best of all the heuristics tested and has actually widened its lead over the others. Surprisingly, the random heuristic moved from fourth best to second best. "This shift is likely due to the rather arbitrary manner in which the sample size parameter was determined, although it may also indicate a decline in effectiveness for the other heuristics." (Dannenbring, 1977). Heuristic RACS dropped from second to fourth position, while heuristic CDS remained in its third position. Heuristics SO and RA follow in that order.

The computation times obtained showed heuristic SO to involve the least computation followed by heuristics RA, RACS, CDS, RAES and R in that order.

"Of major significance is the fact that although quite large differences exist among the average times, the total time involved is quite

small, even for the most costly algorithms on the largest problems.

Both the number of jobs and the number of machines have a significant effect on computation time; the degree of significance is dependent on the nature of the algorithm.

In general, the most economical procedures were the single-shot algorithms that quickly generate only one solution to the problem". (Dannenbring, 1977).

Unfortunately, two results only could be found with respect to the worst-case performance of flow- and job-shop heuristics. Before stating these two results we have the following definition. Let C^*_{max} be an optimum solution to a given flow- or job-shop problem with $m > 2$. Let C_{max} be the completion time of any schedule for the same problem. Then $C_{max}/C^*_{max} \leq m$ (Gonzalez & Sahni, 1978). This worst-case bound of m for schedules in a flow-shop can be reduced to $m/2$ by using the following heuristic H (Gonzalez & Sahni, 1978). Divide the m machines in $m/2$ groups, each group containing at most two machines. The machines in group i are the (2i-1)'st and 2i'th ones. Johnson's algorithm is used to find an optimal schedule for each of the $m/2$ two-machine problems. These $m/2$ optimal schedules are then concatenated to obtain a schedule for the original flow-shop problem. This heuristic requires $O(mn \log n)$ steps. Thus, if C^H_{max} is the completion time of a flow-shop schedule obtained using heuristic H above and C^*_{max} is the optimum solution to the same flow-shop problem, then $C^H_{max}/C^*_{max} \leq \lceil m/2 \rceil$ (Gonzalez & Sahni, 1978).

Remark

With regard to the $\bar{C}$ (mean completion time) criterion, Gonzalez and Sahni (Gonzalez & Sahni, 1978) proved the following results. Let C^* be an optimum solution for a flow- or job-shop problem and $\bar{C}$ be the solution obtained when using any schedule for the same problem. Then $\bar{C}/\bar{C}^* \leq n$. Also, if $\bar{C}_{SPT}$ is the solution obtained when the jobs are ordered according

to the SPT rule (i.e. by ordering the jobs in a non-decreasing order of their sum of processing times), then $\bar{C}_{SPT}/\bar{C}^{*} \leq m$.

8.3 Open- and Job-Shop Problems

In this final section we shall give a brief discussion of the open- and job-shop problems. The problem can be stated as follows. There are n jobs numbered 1,...,n and m machines numbered 1,...,m. Each machine k (k=1,...,m) can process not more than one job at a time. Each job i consists of a set of m_i operations $\{0_{i1},...,0_{im_i}\}$. Each operation corresponds to the processing of job i on some machine for an uninterrupted period of time known as the processing time of job i on that machine. The problem is an *open-shop* if each job consists of a set of operations $\{0_{i1},...,0_{im}\}$, but the order in which these operations are executed is immaterial. If, on the other hand, each job has a specified sequence of operations which may differ from the sequence of operations of other jobs, the problem is a *job-shop*.

With regard to the open-shop problem, we have the following. The $02//C_{max}$ problem can be solved using the algorithm of Gonzalez and Sahni (Gonzalez & Sahni, 1976; Graham et al., 1979). This algorithm requires 0(n) steps. However,there is a little hope that any other open-shop problem can be solved in polynomial time. In fact, the $02/r_i/C_{max}$, $02/tree/C_{max}$ (where the precedence constraints are defined as in Rinnooy Kan, 1976 and Lenstra, 1977: An arc (i,j) in G implies that job i must be completed on all machines before job j can start on the first one) and the $Om//C_{max}$ problems have already been proved to be NP-hard (Lawler et al., 1981A; Lenstra, 1981). The $03//C_{max}$ problem has been proved to be NP-hard also (Gonzalez & Sahni, 1976).

With regard to the job-shop problem, we have the following. There exists an 0(n log n) algorithm (an extension of Johnson's algorithm for the

$F2//C_{max}$ problem) for solving the $J2/m_i \leq 2/C_{max}$ problem (Jackson, 1956), but two minor extensions of this problem, the $J2/m_i \leq 3/C_{max}$ and the $J3/m_i \leq 2/C_{max}$ problems have been proved to be NP-hard (Lenstra, 1977). In fact, even the $J2/m_i \leq 3,\ 1 \leq p_{ij} \leq 2/C_{max}$ and the $J3/m_i \leq 2,\ p_{ij}=1/C_{max}$ problems are NP-hard also (Graham et al., 1979).

"Even within the class of NP-complete problems, the general $Jm//C_{max}$ problem appears to be a very difficult one. A classical and by now traditional quotation from (Conway et al., 1967) asserts pessimistically that 'many proficient people have considered this problem, and all have come away essentially empty-handed. Since this frustration is not reported in the literature, the problem continues to attract investigators who just cannot believe that a problem so simply structured can be so difficult until they have tried it'." (Lenstra, 1977).

An indication of the hardness of this general job-shop problem is clear by the fact that a ten job $J10//C_{max}$ problem formulated in 1963 (Muth & Thompson, 1963), still has not been solved.

CHAPTER NINE

THE TWO-MACHINE FLOW-SHOP PROBLEM UNDER PRECEDENCE CONSTRAINTS

9.1 Introduction

The problem considered in this chapter may be stated as follows. *Consider n jobs (numbered $1,\ldots,n$) and two machines. Each of the two machines can process not more than one job at a time. Each job i has to be processed on machines 1 and 2 in that order during uninterrupted times a_i and b_i respectively. Precedence constraints between jobs are represented by a directed acyclic graph G, where the vertices of G represent the jobs. Job i must be processed before job j on each machine if there exists a directed path from vertex i to vertex j. The objective is to find a schedule that minimizes the maximum completion time on the second machine.*

Given any sequence $\pi = \{\pi(1),\ldots,\pi(n)\}$, the minimum completion times $C'_{\pi(1)}$ and $C_{\pi(1)}$ of the first job in the sequence on the first and second machines are equal to $a_{\pi(1)}$ and $a_{\pi(1)} + b_{\pi(1)}$ respectively. The minimum completion times of any other job $\pi(i)$ $(i=2,\ldots,n)$ on the first and second machines are given by $C'_{\pi(i)} = C'_{\pi(i-1)} + a_{\pi(i)}$ and $C_{\pi(i)} = \max(C_{\pi(i-1)}, C'_{\pi(i)}) + b_{\pi(i)}$ respectively.

We recall from Section 8.1 that the two problems $F2/\beta/\gamma$ and $P2/\beta/\gamma$ are equivalent, i.e. we only need to consider schedules in which the same processing order occurs on both machines.

Johnson (Johnson, 1954) gave an efficient algorithm for the unconstrained case, which is considered as one of the most important breakthrough in machine scheduling problems.

Mitten (Mitten, 1959A and 1959B) considered a problem which is similar to Johnson's problem. In his model, each job i has processing times a_i and b_i on the first and second machines respectively, a non-negative start-lag a'_i and a non-negative stop-lag b'_i. The start lag is

defined to be the minimum time between the start of processing job i on the first machine and the start of processing job i on the second machine, while the stop lag is defined to be the minimum time between the completion of processing job i on the first machine and the completion of job i on the second machine. He gave a decision rule to obtain a processing order of the jobs on both machines in order to minimize the total elapsed time.

Kurisu (Kurisu, 1976) applied Mitten's results to provide an efficient algorithm for the two machine flow shop problem under precedence constraints in which the constraints form a "parallel chain". This algorithm is based on forming composite jobs, each of which consists of at least one job that must be processed without interruption in the same order they form that composite job.

Sidney (Sidney, 1979) and Monma (Monma, 1979) applied Kurisu's results to provide an efficient algorithm for the two machine problem with series-parallel constraints. The algorithm requires $O(n \log n)$ steps. The problem has also been considered in (Monma & Sidney, 1979).

However, for general precedence constraints, the problem has been shown to be NP-hard (Monma, —). Kurisu (Kurisu, 1977) studied this general case and gave an effection branch and search algorithm to obtain an optimum sequence. He did not make any attempt to derive a lower bounding procedure to be used in his proposed algorithm.

We now introduce some terms that are used in later sections. The *transitive closure* of the directed graph G is the graph obtained by adding all arcs (i,j) (if it is not in G) to G whenever there is a directed path from vertex i to vertex j. The *transitive reduction* of G is the graph obtained by deleting all arcs (i,j) from G whenever there is a directed path from vertex i to vertex j other than the arc (i,j) itself. The *inverse of* G is the graph obtained by reversing the direction of every arc (i,j) in G. The *adjacency matrix* of the transitive closure of G is

the $n \times n$ matrix $X = (x_{ij})$, where $x_{ij}=1$ if there exists an arc (i,j) in the transitive closure of G and $x_{ij}=0$ otherwise. Job i is a *predecessor* of job j and job j is a *successor* of job i if the arc (i,j) exists in the transitive closure of G. Job i is a *direct predecessor* of job j and job j is a *direct successor* of job i if the arc (i,j) exists in the transitive reduction of G.

In this chapter we shall give a bounding procedure to solve this general case (i.e. $F2/prec/C_{max}$) using the branching rule proposed by Kurisu (Kurisu, 1977). Section 9.2 contains Kurisu's branching rule together with some dominance theorems. Our bounding procedures will be given in Section 9.3 followed by a heuristic in Section 9.4. Our branch and bound algorithm will be explained in Section 9.5, where we shall consider an example from (Kurisu, 1977). A complete description of the algorithm is given in Section 9.6. Computational experience is presented in Section 9.7 which is followed by some concluding remarks in Section 9.8.

9.2 Branching Rule and Dominance

We start this section by giving the branching rule proposed by Kurisu (Kurisu, 1977). This branching rule partitions the set of feasible solutions to the problem into subsets, some of which will be eliminated using the dominance theorems to be given below. Essentially, at each branching a job is sequenced either first, last, immediately before another given job or immediately after another given job.

We now give four results that were used by Kurisu to act as dominance rules to reduce the number of branches of the search tree. The theorems are results for the original problem, where the corollaries are the corresponding results for the equivalent inverse problem in which the objective is to minimize the maximum completion time subject to the precedence constraints defined by the inverse graph. It has become clear now that the problem is symmetric.

Let us first define some notations that are going to be used below. Let S denote the set of unscheduled jobs, let B and A be the set of jobs with no predecessors and the set of jobs with no successors respectively. Define B_i and A_i to be the set of jobs that must precede and succeed job i in any feasible schedule respectively. Also, define B_i' and A_i' to be the set of jobs that directly precede and directly succeed job i.

Finally, we like to point out that all jobs i to be considered in this chapter will have the property that $a_i'=C_i-b_i$ and $b_i'=C_i-a_i$, where a_i' and b_i' are as defined in the previous section and C_i is the minimum completion time of job i, as illustrated in Figure 9.1. The original problem is obtained by setting $a_i'=a_i$ and $b_i'=b_i$ for all i. As we shall see below, it will be useful to substitute jobs for sequences of jobs that are known to be processed without interruption. If $K=(k_1,k_2,\ldots,k_J)$ is a composite job consisting of J single jobs, then $a_K'=C_K-b_K$ and $b_K'=C_K-a_K$, where $a_K = \sum_{i=1}^{J} a_{ki}$, $b_K = \sum_{i=1}^{J} b_{ki}$ and C_K is the minimum possible completion time of the composite job K (assuming that only jobs in K are to be processed). Figure 9.1 with K substituted for i, illustrates a composite job. We remark that

$$C_K = \max_{h=1,\ldots,J} \left(\sum_{i=1}^{h-1} a_{ki} + C_h + \sum_{i=h+1}^{J} b_{ki} \right)$$

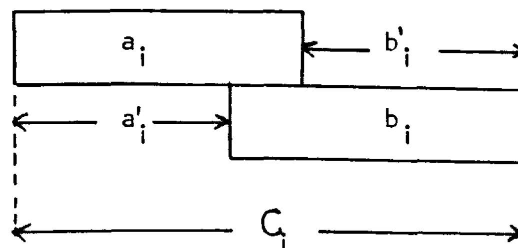


Figure 9.1: Typical job

Theorem 9.1 (Kurisu, 1977)

If for a job $i\varepsilon B$, $a_i' \leq b_i'$ *and* $a_i' \leq a_j'$ *for all jobs* $j\varepsilon B$, *then there exists an optimum schedule in which job* i *is sequenced first.*

Corollary 9.1 (Kurisu, 1977)

If for a job $i\varepsilon A$, $b_i' \leq a_i'$ *and* $b_i' \leq b_j'$ *for all jobs* $j\varepsilon A$, *then there exists an optimum schedule in which job* i *is sequenced last.*

Theorem 9.2 (Kurisu, 1977)

If for a job i *with* $B_i \neq \emptyset$, $a_i' \leq b_i'$ *and* $a_i' \leq a_j'$ *for all jobs* $j\varepsilon S$, *then there exists an optimum schedule in which job* i *is sequenced immediately after one of its direct predecessors.*

Corollary 9.2 (Kurisu, 1977)

If for a job i *with* $A_i \neq \emptyset$, $b_i' \leq a_i'$ *and* $b_i' \leq b_j'$ *for all jobs* $j\varepsilon S$, *then there exists an optimum schedule in which job* i *is sequenced immediately before one of its direct successors.*

We next give a formal statement of Kurisu's branching procedure.

Step 1.1: If only one job exists in B, sequence this job first; otherwise proceed to Step 1.2.

Step 1.2: If only one job exists in A, sequence this job last; otherwise proceed to Step 2.1.

Step 2.1: If there exists a job $i\varepsilon B$ such that $a_i' \leq b_i'$ and $a_i' \leq a_j'$ for all $j\varepsilon B$, then sequence job i first (Theorem 9.1); otherwise proceed to Step 2.2.

Step 2.2: If there exists a job $i\varepsilon A$ such that $b_i' \leq a_i'$ and $b_i' \leq b_j'$ for all $j\varepsilon A$, then sequence job i last (Corollary 9.1); otherwise proceed to Step 3.1.

Step 3.1: If there exists a job $i\varepsilon B$, such that $a_i' \leq b_i'$ and $a_i' \leq a_j'$ for all jobs, then let n_1 be the number of jobs in B_i'; otherwise let $n_1 = n$.

Step 3.2: If there exists a job $i'\varepsilon A$, such that $a_i' > b_i'$, and $b_i' \leq b_j'$ for all jobs j, then let n_2 be the number of jobs in A_i'; otherwise let $n_2 = n$.

Step 4.1: If $0 < n_1 < n_2$, a composite job (i.e. a new vertex) k= j i is added to G, vertex j and vertex i are deleted from G where $j\varepsilon B_i'$; otherwise proceed to Step 4.2.

Step 4.2: A composite job $k = i'\,j$ is added to G, vertex i' and vertex j are deleted from G, where $j \in A'_{i'}$.

Whenever a new composite job $K = i\,j$ is performed, the precedence graph G is updated as follows.

(a) Vertex i and vertex j are deleted and a new single vertex $K=(i,j)$ is added.

(b) For each arc (h,i) or (h,j) in G, where $h \neq i$, an arc (h,K) is added.

(c) For each arc (i,h) or (j,h) in G, where $h \neq j$, an arc (K,h) is added.

Now we shall give two results which will be referred to as the dominance rules. Clearly, dominance rules are particularly useful when a node can be eliminated which has a lower bound that is less then the optimum solution.

Let L denote the value of any lower bound.

Theorem 9.3 (Potts, 1974)

If for a job $i \in B$, $a_i \leq b_i$ and $a_i + \sum_{j=1}^{n} b_j \leq L$, then there exists an optimum sequence in which job i is sequenced first.

Corollary 9.3 (Potts, 1974)

If for a job $i \in A$, $b_i \leq a_i$ and $b_i + \sum_{j=1}^{n} a_j \leq L$, then there exists an optimum sequence in which job i is sequenced last.

9.3 Lower Bounds

In this section, we shall be interested in deriving lower bounds on the maximum completion time for all feasible schedules beginning with an initial partial sequence σ_1 and ending with a final partial sequence σ_2. Let S_1 be the set of jobs sequenced in σ_1 and S_2 be the set of jobs sequenced in σ_2. Also, let $C_{\sigma 1}$ denote the minimum completion time of all jobs in σ_1 and $C'_{\sigma 2}$ denote the minimum time between the start of processing jobs in σ_2 on the first machine and the completion of processing jobs in σ_2 on the second machine (we define $C_{\sigma 1} = 0$ if $S_1 = \emptyset$ and $C'_{\sigma 2} = 0$ if $S_2 = \emptyset$). Finally, for each job i we define $I_i = \{j/(i,j)$ and (j,i) are not in $G\}$.

9.3.1 Job Based Bound

Lower bounds based on individual jobs (job-based bound) rather than individual machines (machine-based bound) were first proposed by McMahon and Burton (McMahon & Burton, 1967).

Consider an unscheduled job i. Each job $j=1,\ldots,n, j\neq i$ must be sequenced either before or after job i. If job j precedes job i either because $j\epsilon S_1$ or because an arc (j,i) exists in G, then a_j must be added when computing a lower bound. If job j succeeds job i, either because $j\epsilon S_2$ or an arc (i,j) exists in G, then b_j can be added to the lower bound. If, on the other hand, it is not known whether job j precedes or succeeds job i, then the smaller of a_j and b_j may be added to the lower bound. Finally, the minimum completion time of job i (ignoring other jobs), C_i, can also be added to give a realistic bound LB_i (see Figure 9.2) as follows:

$$LB_i = \sum_{j\epsilon S_1} a_j + \sum_{j\epsilon B_i} a_j + C_i + \sum_{j\epsilon S_2} b_j + \sum_{j\epsilon A_i} b_j + \sum_{j\epsilon I_i} \min(a_j,b_j)$$

We shall refer to such a job i as the *critical job*.

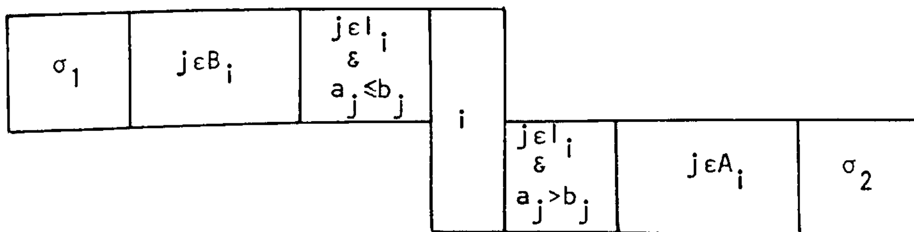


Figure 9.2: The structure of the proposed job based bound

Thus an overall lower bound is given by

$$LB = \max_{i\epsilon S} (LB_i)$$

Two additional lower bounds, based on σ_1 and σ_2 instead of job i, may be given as follows:

$$LB_{\sigma 1} = C_{\sigma 1} + \sum_{j \notin S_1} b_j, \qquad \text{if } S_1 \neq \emptyset.$$

$$= \min_{j \in B} a_j + \sum_{j=1}^{n} b_j, \qquad \text{if } S_1 = \emptyset.$$

and

$$LB_{\sigma 2} = C'_{\sigma 2} + \sum_{j \notin S_2} a_j, \qquad \text{if } S_2 \neq \emptyset.$$

$$= \min_{j \in A} b_j + \sum_{j=1}^{n} a_j, \qquad \text{if } S_2 = \emptyset.$$

We define

$$LB_{JB} = \max(LB_{\sigma 1}, LB, LB_{\sigma 2})$$

9.3.2 Conflict Bound

Let LB_i $(i \varepsilon S)$ denote the lower bounds obtained as above. Consider two jobs i and j in S such that the two arcs (i,j) and (j,i) are not in G. We have two cases to look at.

Case 1: $a_j > b_j$ when $a_i > b_i$

If job i is chosen to act as the critical job, then b_j will be added when computing LB_i (i.e. as if job j is sequenced after job i). When job j is chosen to act as the critical job, then b_i will be added when computing LB_j (i.e. as if job i is sequenced after job j). But since we can have either i before j $(i \to j)$ or i after j $(j \to i)$, then we have one of two ways in which we may be able to improve the lower bound.

(a) $i \to j$

In this case LB_j can be increased by $a_i - b_i$. Thus a lower bound for this case is given by:

$$\max(LB_i, LB_j + a_i - b_i)$$

(b) $j \rightarrow i$

In this case LB_i can be increased by $a_j - b_j$. Thus, a lower bound for this case is given by:

$$\max(LB_i + a_j - b_j, LB_j)$$

Thus a lower bound for the problem is given by:

$$LB_{ij} = \min\{\max(LB_i, LB_j + a_i - b_i), \max(LB_i + a_j - b_j, LB_j)\}$$

Case 2: $a_j < b_j$ when $a_i < b_i$

Similar to Case 1, a lower bound is given by:

$$LB_{ij} = \min\{\max(LB_i, LB_j - a_i + b_i), \max(LB_i - a_j + b_j, LB_j)\}$$

An overall bound LB' is given by:

$$LB' = \max_{i,j}\{LB_{ij} / i \text{ and } j \text{ as defined above}\}$$

As we have said before, i and j can be any two jobs in S such that the two arcs (i,j) and (j,i) do not belong to G. It is clear that LB_{ij} is likely to be increased when LB_i and LB_j (or both) are as large as possible. For this reason and to reduce computational requirements, we propose to choose one of the jobs, say i, with $LB_i = LB$ and then compute LB_{ij} for all $j \in S$ such that arcs (i,j) and (j,i) are not in G.

It is clear that there is no need to compute a similar conflict bound LB_{ij} when $a_i < b_i$ and $a_j > b_j$ since in this case $LB_{ij} = \max(LB_i, LB_j)$. Also, there is no need to compute LB_{ij} if there exists an arc between vertex i and vertex j ((i,j) or (j,i)) in G, since there is only one choice to take that is according to the precedence graph G and hence $LB_{ij} = \max(LB_i, LB_j)$.

From the above we conclude that the conflict bound, LB', is at least as good as the job based bound.

We define

$$LB_{CB} = \max(LB_{\sigma 1}, LB', LB_{\sigma 2})$$

9.4 Heuristic

It is well-known that computation can be reduced by using a heuristic to find a good solution to act as an upper bound. The heuristic proposed below is applied once at the top of the search tree. It requires $0(n^2)$ if the transitive closure of the directed graph G is known.

Follow Steps 1 and 2 of the formal statement of the branching procedure given in Section 9.2.

Step 3: If $a_i > b_i$ for all $j \varepsilon B$ and $a_j \leq b_j$ for all $j \varepsilon A$ then sequence a job $i \varepsilon A$ with $\min(a_j, b_i) \leq \min(a_i, b_j)$ for all jobs $j \varepsilon A$ last and go to Step 5; otherwise proceed to Step 4.

Step 4: Sequence a job $i \varepsilon B$ with $\min(a_i, b_j) \leq \min(a_j, b_i)$ for all jobs $j \varepsilon B$ first and proceed to Step 5.

Step 5: Delete job i from G and update the two sets B and A. If all jobs have been sequenced, stop; otherwise go to Step 1.

If π is the sequence obtained using the above procedure, then the completion of each job sequenced in π can be computed. The completion time of the last job in the sequence forms an upper bound on the value of C_{max}.

Remark

One can obtain a sequence which is at least as good as the sequence obtained using the above heuristic as follows. Consider the branching procedure given in Section 9.2. Suppose that h nodes (each node corresponds to a composite job been performed according to Step 4 of the algorithm, and where $h = \min(n_1, n_2)$ exist at level k of the search tree. Apply the heuristic given above at every one of these h nodes. A node with the smallest value of the heuristic is chosen to branch from. All other nodes are eliminated from the search tree.

This heuristic method is given in Sections 3.3.3 and 5.5.

9.5 Example

In this section we shall illustrate our branch and bound procedure

by considering an example which appeared in Kurisu (Kurisu, 1977). The processing times of the jobs on the two machines are given in Table 9.1. The precedence graph G for the example is given in Figure 9.3. Each node has three entries: job number i (top), a start-lag a_i' (left) and a stop-lag b_i' (right).

Table 9.1

i	1	2	3	4	5	6	7	8	9
a_i	4	6	3	8	10	5	9	2	3
b_i	7	5	1	4	7	6	3	9	4

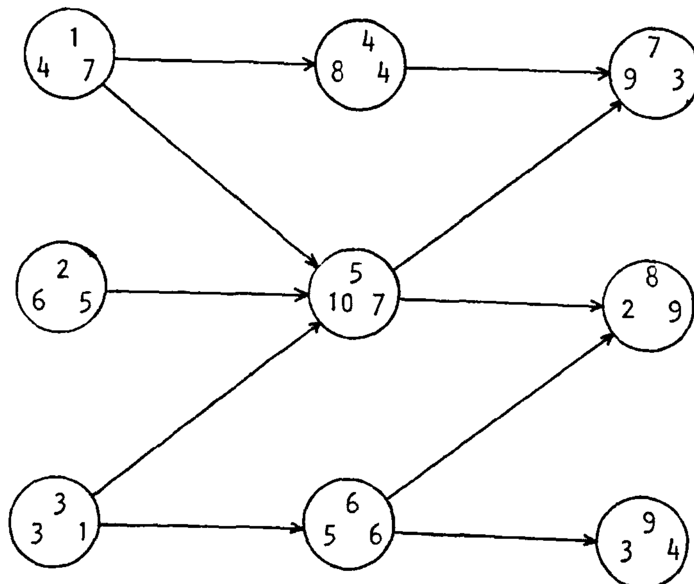


Figure 9.3: Precedence Graph G

1. We have:

$B = \{1,2,3\}$ and $A = \{7,8,9\}$

Since $b'_7 \leq \min(a'_7, b'_8, b'_9)$, then job 7 can be sequenced last (Corollary 9.1).

2. Having sequenced job 7 last (i.e. $\sigma_2 = \{7\}$) and deleted node 7 and all arcs to that node in G (let G_1 be the resulting graph), the set of jobs with no successors A becomes $A = \{4,8,9\}$.

Since $b'_4 \leq \min(a'_4, b'_8, b'_9)$, then job 4 can be sequenced last. Deleting node 4 and all arcs to that node in G_1, we get the precedence graph G_2 which is shown in Figure 9.4.

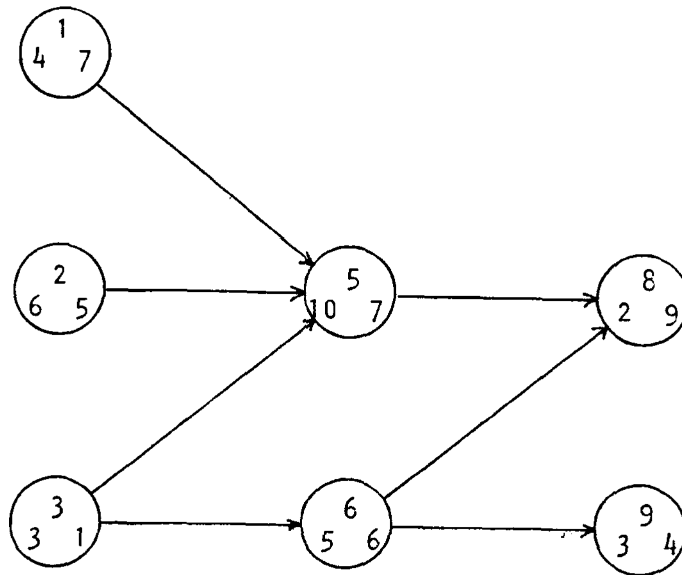


Figure 9.4: Precedence Graph G_2

3. We have

$\sigma_2 = \{4,7\}$, $B = \{1,2,3\}$ and $A = \{8,9\}$

Both Theorem 9.1 and Corollary 9.1 cannot be satisfied here and thus composite jobs have to be formed.

Set of jobs with at least one predecessor is {5,6,8,9}. Since $a_8' \leq \min(b_8', a_5', a_6', a_9')$, set $i=8$. We have $B_i' = \{5,6\}$ and thus $n_1=2$.

Set of jobs with at least one successor is {1,2,3,5,6}. Since $b_3' \leq \min(a_3', b_1', b_2', b_5', b_8')$, set $i' = 3$. We have $A_3' = \{5,6\}$ and thus $n_2=2$.

Since $n_1 = n_2$ then we have two composite jobs to consider: (3-5) and (3-6).

We shall compute our lower bounds for the node corresponding to forming a composite job (3-5). The precedence graph for this case, G_3, is given in Figure 9.5.

$$\text{Since } S_1 = \emptyset, \quad \therefore LB_{\sigma 1} = \min_{i \in B} a_i + \sum_{i=1}^{n} b_i = 4 + 46 = 50$$

and

$$\sigma_2 = (4,7), \quad \therefore LB_{\sigma 2} = C_{\sigma 2}' + \sum_{j \notin S_2} a_j = 21+4+6+8+10+2+3 = 54$$

We remark that $a_{(3-6)} = 8$, $b_{(3-6)} = 7$ and $C_{(3-6)} = 14$.

$$LB_1 = (4+7) + (4+3) + (7+9) + (5+7+3) = 49.$$

$$LB_2 = (6+5) + (4+3) + (7+9) + (4+7+3) = 48.$$

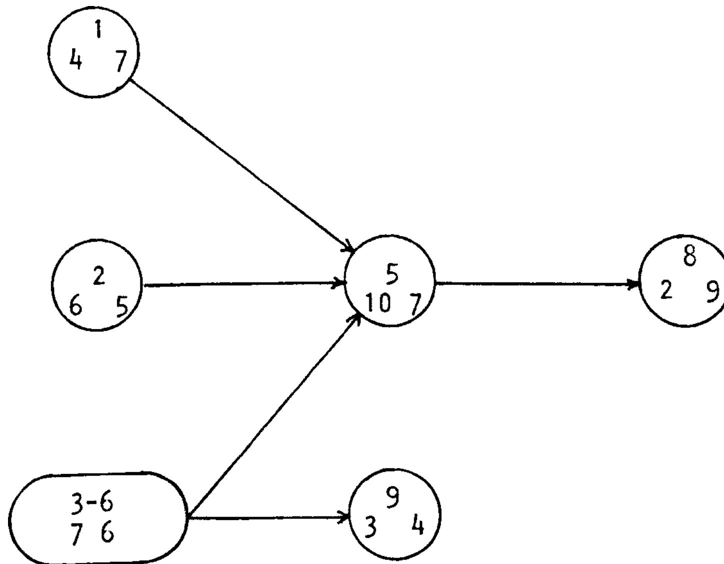


Figure 9.5: Precedence Graph G_3

$LB_{(3-6)} = (14) + (4+3) + (4+7+9) + (4+5) = 50.$

$LB_5 = (4+6+8) + (10+7) + (4+3) + (9) + (3) = 54.$

$LB_8 = (4+6+8+10) + (2+9) + (4+3) + (3) = 49.$

$LB_9 = (8) + (3+4) + (4+3) + (4+5+7+2) = 40.$

$LB = 54$

$\varepsilon LB_{JB} = \max(50,54,54) = 54.$

A conflict bound $LB_{1,9}$ can be computed as follows:

$LB_{1,9} = \min(\max(49,40-4+7), \max(49-3+4,40))$

$= 48$

Similarly, $LB_{8,9} = 49$ and $LB_{2,3-6} = 50$.

In this case we have $LB' = LB = 54$ and hence $LB_{CB} = LB_{JB} = 54$.

In a similar way, one can compute lower bounds for the node corresponding to forming a composite job (3-5) to give:

$LB_{CB} = LB_{JB} = 56.$

4. We have

$B = \{1,2,(3-6)\}$ and $A = \{8,9\}$.

Since $a_1' \leq \min(b_1', a_2', a'_{3-6})$. Thus job 1 can be sequenced first (Theorem 9.1). Deleting node 1 and all arcs from that node in G_3, we get the precedence graph G_4 given in Figure 9.6.

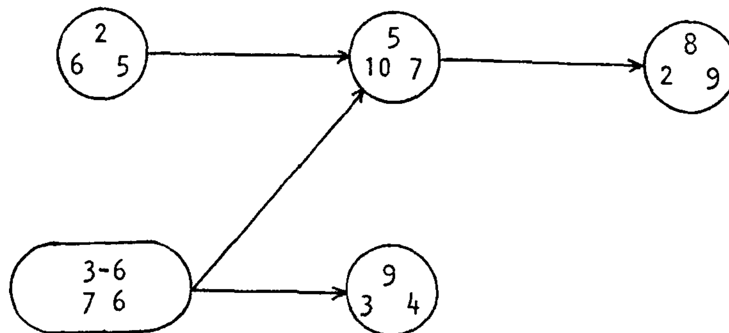


Figure 9.6: Precedence Graph G_4

The procedure can be completed in a similar way to give an optimum sequence for the example (1,3,6,9,2,5,8,4,7), which has a value 54.

9.6 The Algorithm

Here, we shall give a complete description of the algorithm in its general form, i.e. in the case where the heuristic, dominance rules, the job based bound and conflict job bound are all used. Each of the other algorithms is a special case of the algorithm described here.

We start the algorithm by computing the transitive closure of the precedence graph. This requires $0(n^3)$ steps. We then apply the heuristic method given in Section 9.4 to obtain a sequence. The completion time of each job in this sequence is then calculated. The value of the completion time of the last job in this sequence forms an initial upper bound on C_{max}.

The branch and bound procedure is then started. Before any new node is created, Steps 1 and 2 of Section 9.2 and the dominance rules (Theorem 9.3 and Corollary 9.3) are checked in that order. If a job i can be found satisfying the conditions of Step 1.1, Step 2.1 or Theorem 9.3 (Step 1.2, Step 2.2 or Corollary 9.3), then a single successor node is created in the search tree, with a lower bound equal to that of its parent, corresponding to job i being sequenced first (last).

For each node of the search tree (corresponds to performing a composite job k=ij according to Step 3 or 4 of Section 9.2), the transitive closure of G is updated by adding the arc (h,j) whenever an arc (h,i) exists (h≠j) and by adding the arc (h,k) whenever the arcs (h,i) and (j,k) exist (h≠j, k≠i). The lower bounding procedure for that node is then started as follows. An 0(n) steps is spent on computing LB_i for each job i (i=1,...,n), i.e. $0(n^2)$ steps is needed to compute, LB, the job based bound of Section 9.3.1. If LB is not less than the current upper bound, this node is eliminated. Otherwise, 0(n) steps is needed to compute $LB_{\sigma 1}$ and

$LB_{\sigma 2}$ of Section 9.3.1. If the lower bound obtained so far is not less than the current upper bound, this node is eliminated. Otherwise, given a job i with $LB_i = LB$, we spend a further O(n) steps on calculating LB', the conflict bound. This node is eliminated if LB' is not less than the upper bound, otherwise it forms the basis for our next branchings.

The branch and bound procedure continues in a similar way. Whenever a complete sequence is obtained, this sequence is evaluated and the upper bound is altered if the new value is less than the old one.

Finally, our search strategy is given. A node from which to branch is chosen at random from the most recently created subset of nodes. As mentioned in Section 3.2, the advantage of this type of search strategy is that it requires less storage space than if another search strategy is used.

9.7 Computational Experience

9.7.1 Algorithm Representation

It is clear from the above sections that each algorithm to be considered can be represented by (LBD, UBD, DOM) where:

LBD = JB, CB or -	Describes the bound to be used (see Section 9.3). If neither of the bounds is used LBD = -.
UBD = H or -	According to whether the heuristic of Section 9.4 is or is not used.
DOM = D or -	According to whether the dominance rules (Theorem 9.3 and Corollary 9.3) of Section 9.2 are or are not used.

9.7.2 Test Problems

The algorithms were tested on problems with 20, 40, 50 and 60 jobs (initial tests showed problems with 60 jobs to be much harder than problems with 40 jobs. For this reason we decided to include results for 50 job problems). These problems contained problems with random and correlated

processing times. For each job i, two integer processing times a_i and b_i were generated from the uniform distribution [1,100] or $[20\varepsilon_i+1, 20\varepsilon_i+20]$ according to whether the processing times for that problem are to be random or correlated, where ε_i is an integer randomly drawn from {1,2,3,4,5}. This method of processing times generation follows that of (Lageweg, Lenstra & Rinnooy Kan, 1978). In the precedence graph G, each arc (i,j) with i<j was included with a given probability p. For each value of n, twenty problems (ten with random processing times and ten with correlated processing times) were generated for each of the p values 0.05, 0.2, 0.3, 0.5 and 0.75. Thus 400 problems in all were used to test the algorithms.

9.7.3 Computational Results

The algorithms were coded in FORTRAN IV and run on a CDC 7600 computer.

Computational results for problems with random processing times are given in Tables 9.2 to 9.4. Computational results for problems with correlated processing times are given in Tables 9.5 to 9.7. Whenever a problem was not solved within the time limit of 70 seconds or after 15,000 nodes had been generated (whichever occurs first), computation was abandoned for that problem. Thus, in some cases the figures given in Tables 9.2, 9.3, 9.5 and 9.6 will be lower bounds on average computation times and average number of nodes.

As we mentioned above, the test problems have been divided into two groups, the first group contains problems with random processing times and the second one contains problems with correlated processing times.

With regard to the first group, average computation time, average number of nodes and number of unsolved problems are given in Tables 9.2, 9.3 and 9.4 respectively. The first three columns of each Table compare the performance of Kurisu's branching rule, with the job based bound and

Table 9.2: Average Computation Time for Problems with Random Processing Times**

n	p	ALGORITHM (-,-,-)	(JB,-,-)	(CB,-,-)	(CB,H,-)	(CB,H,D)
20	0.05	0.03	0.03	0.03	0.03	0.04
	0.2	0.06	0.05	0.05	0.05	0.05
	0.3	0.07	0.06	0.06	0.05	0.05
	0.5	0.07	0.06	0.05	0.06	0.06
	0.75	0.05	0.06	0.05	0.06	0.06
40	0.05	0.35	0.30	0.30	0.28	0.33
	0.2	19.10*	3.95	3.62	3.58	3.31
	0.3	13.56*	2.00	1.97	1.91	1.70
	0.5	3.71	0.87	0.87	0.83	0.80
	0.75	0.77	0.73	0.73	0.71	0.71
50	0.05	5.01	0.67	0.67	0.58	0.65
	0.2	--	22.63*	11.89	11.76	11.82
	0.3	40.65*	8.23	4.35	4.29	3.72
	0.5	9.48*	2.42	2.42	2.32	2.21
	0.75	1.99	1.77	1.77	1.67	1.68
60	0.05	41.12*	1.27	1.26	1.08	1.13
	0.2	--	--	--	--	--
	0.3	38.42*	38.54*	38.42*	38.24*	37.78*
	0.5	24.10*	3.93	3.93	3.71	3.70
	0.75	4.33	3.65	3.64	3.43	3.44

** Times are in CPU seconds.

* Lower bounds because of unsolved problems.

- More than 7 problems were left unsolved.

Table 9.3: Average Number of Nodes for Problems with Random Processing Times

n	p	ALGORITHM (-,-,-)	(JB,-,-)	(CB,-,-)	(CB,H,-)	(CB,H,D)
20	0.05	4	3	3	1	0
	0.2	28	10	10	7	5
	0.3	33	12	12	5	5
	0.5	17	7	7	5	4
	0.75	2	2	2	1	0
40	0.05	36	13	13	1	1
	0.2	7915*	935	816	800	772
	0.3	5291*	371	358	335	300
	0.5	1109	40	40	29	23
	0.75	25	8	8	2	1
50	0.05	1595	24	24	4	3
	0.2	--	3669*	1525	1492	1446
	0.3	11335*	1480	606	594	449
	0.5	2144*	150	150	133	109
	0.75	68	15	15	1	1
60	0.05	10664*	39	39	10	8
	0.2	--	--	--	--	--
	0.3	4884*	5094*	4883*	4861*	4964*
	0.5	4452*	64	64	42	40
	0.75	151	25	24	6	6

* Lower bounds because of unsolved problems.

\- More than 7 problems were left unsolved.

Table 9.4: Number of Unsolved Problems with Random Processing Times

n	p	ALGORITHM (-,-,-)	(JB,-,-)	(CB,-,-)	(CB,H,-)	(CB,H,D)
40	0.05	0	0	0	0	0
	0.2	4	0	0	0	0
	0.3	2	0	0	0	0
	0.5	0	0	0	0	0
	0.75	0	0	0	0	0
50	0.05	0	0	0	0	0
	0.2	>7	2	0	0	0
	0.3	6	0	0	0	0
	0.5	1	0	0	0	0
	0.75	0	0	0	0	0
60	0.05	7	0	0	0	0
	0.2	>7	>7	>7	>7	>7
	0.3	5	5	5	5	5
	0.5	1	0	0	0	0
	0.75	0	0	0	0	0

Table 9.5: Average Computation Time for Problems with Correlated Processing Times**

n	p	(-,-,-)	(JB,-,-)	ALGORITHM (CB,-,-)	(CB,H,-)	(CB,H,D)
20	0.05	0.03	0.03	0.03	0.04	0.04
	0.2	0.16	0.09	0.09	0.09	0.07
	0.3	0.16	0.11	0.10	0.08	0.08
	0.5	0.08	0.06	0.06	0.06	0.06
	0.75	0.05	0.06	0.06	0.06	0.06
40	0.05	4.26	1.05	1.05	1.08	0.69
	0.2	--	4.21	4.16	3.93	2.45
	0.3	14.28*	0.97	0.97	0.91	0.88
	0.5	1.40	0.81	0.81	0.75	0.72
	0.75	0.77	0.76	0.76	0.72	0.72
50	0.05	17.36*	12.13*	12.22*	12.28*	8.16*
	0.2	--	12.19*	12.10*	12.07*	11.63*
	0.3	--	15.96*	13.63*	13.63*	13.11*
	0.5	11.46*	6.45	6.45	6.34	3.62
	0.75	1.97	1.74	1.74	1.68	1.69
60	0.05	52.42*	37.37*	37.36*	37.40*	29.31*
	0.2	--	--	--	--	--
	0.3	50.06*	50.06*	50.06*	49.95*	46.29*
	0.5	28.92*	19.07*	17.63*	17.49*	12.37*
	0.75	4.43	3.53	3.52	3.44	3.43

** Times are in CPU seconds.

* Lower bounds because of unsolved problems.

- More than 7 problems were left unsolved.

Table 9.6: Average Number of Nodes for Problems with Correlated Processing Times

n	p	ALGORITHM (-,-,-)	(JB,-,-)	(CB,-,-)	(CB,H,-)	(CB,H,D)
20	0.05	4	4	4	4	2
	0.2	112	40	36	30	18
	0.3	113	43	41	20	20
	0.5	28	9	9	5	3
	0.75	3	2	2	1	1
40	0.05	1698	210	209	208	39
	0.2	--	902	885	811	450
	0.3	5838*	99	99	81	70
	0.5	267	28	28	12	8
	0.75	25	12	12	2	1
50	0.05	5671*	2767*	2767*	2767*	1390*
	0.2	--	1740*	1714*	1698*	1367*
	0.3	--	2370*	1913*	1905*	1613*
	0.5	2626*	748	747	726	281
	0.75	77	14	14	1	1
60	0.05	12454*	4589*	4583*	4580*	2796*
	0.2	--	--	--	--	--
	0.3	6626*	6812*	6629*	6615*	5095*
	0.5	5326*	2340*	2072*	2050*	1378*
	0.75	177	17	17	5	3

* Lower bounds because of unsolved problems.

- More than 7 problems were left unsolved.

Table 9.7: Number of Unsolved Problems with Correlated Processing Times

n	p	ALGORITHM (-,-,-)	(JB,-,-)	(CB,-,-)	(CB,H,-)	(CB,H,D)
40	0.05	0	0	0	0	0
	0.2	>7	0	0	0	0
	0.3	3	0	0	0	0
	0.5	0	0	0	0	0
	0.75	0	0	0	0	0
50	0.05	2	1	1	1	1
	0.2	>7	1	1	1	1
	0.3	>7	1	1	1	1
	0.5	1	0	0	0	0
	0.75	0	0	0	0	0
60	0.05	7	5	5	5	3
	0.2	>7	>7	>7	>7	>7
	0.3	7	7	7	7	6
	0.5	2	2	2	2	1
	0.75	0	0	0	0	0

with the conflict bound respectively. The excellent performance of our algorithms (JB,-,-) and (CB,-,-), on this class of problems is clear, especially for problems of size 40, which we managed to solve using either of our bounds. Furthermore, using algorithm (CB,-,-), we even managed to solve all problems of size 50. Unfortunately, the search trees for two problems (out of 50) of size 50 become large when our first algorithm, i.e. (JB,-,-) is used and hence these two problems were left unsolved. Among the test problems of size 60, our algorithms were particularly effective for problems with $p = 0.05$, 0.5 and 0.75, but problems with $p = 0.2$ appear to be too hard for all the algorithms. Columns 2 and 3 of Tables 9.2, 9.3 and 9.4 show that the critical bound (algorithm (CB,-,-)) performs better than the job based bound (algorithm (JB,-,-)) and hence it will be used henceforth.

By adding our upper bounding procedure, Columns 3 and 4 of Tables 9.2 and 9.3 show that a small reduction in computation (except for some problems of size 20) and in the number of nodes can be achieved. Finally, by adding the dominance rules, Columns 4 and 5 of Tables 9.2 and 9.3 show that a further reduction in computation can be achieved for most problems with $p = 0.2$, 0.3 and 0.5 and that number of nodes has also been reduced in most cases.

With regard to the second group, average computation time, average number of nodes and number of unsolved problems are given in Tables 9.5, 9.6 and 9.7 respectively. The excellent performance of our algorithm for problems of size 40 is still clear, especially for problems with $p = 0.2$ which have been solved using either of our two algorithms (JB,-,-) or (CB,-,-), while using Kurisu's algorithm (-,-,-) nearly all the ten problems were left unsolved. For problems of size 50, our algorithms have also performed well, where at most one problem was left unsolved using either of our algorithms, compared to all problems being left unsolved in two

cases (p = 0.2 and 0.3) using algorithm (-,-,-). Algorithms (JB,-,-) and (CB,-,-) have not performed well on problems of size 60, compared with their performance on random problems.

By adding our upper bounding procedure, columns 3 and 4 of Tables 9.5 and 9.6 show that a small reduction in computation and in number of nodes can be achieved in most cases. Finally, by adding the dominance rules, columns 4 and 5 of Tables 9.5, 9.6 and 9.7 show that the effect of these rules have become very clear on this class of problems. They were particularly useful for problems of size 60 and p = 0.05, 0.3 and 0.5.

Disregarding the results for problems of size 60 and p = 0.2, the number of unsolved random and correlated problems using algorithm (CB,H,D) are 5 and 13 respectively. Thus, the correlated problems appear to be the most difficult and the most challenging, which is in accordance with results obtained for problems with no precedence constraints (Lageweg et al., 1978; Potts, 1980A), and with that of Chapter 10 (to follow) for the permutation flow shop problem under precedence constraints.

9.8 Concluding Remarks

All our algorithms showed superiority over Kurisu's algorithm. This superiority is particularly clear when n = 40 or 50 and p = 0.2 or 0.3. However, all our algorithms are satisfactory for solving problems of sizes up to 50 jobs. In fact, using the conflict bound we managed to solve all problems of sizes up to 50 jobs except for three correlated problems with 50 jobs. Unfortunately, when n = 60 and p = 0.2 or 0.3, the problem becomes too hard for all the algorithms.

Although the conflict bound performed very well, it is possible to improve it further. This can be done as follows. Let i and j as defined in Section 9.3.2, and $\max(LB_i, LB_j + a_i - b_i)$ in case 1a, can be written as follows:

$$\max(LB_i,\ LB_j + a_i - b_i + \sum_{\substack{l \in B_i \cap I_j \\ \& a_h > b_h}} (a_h - b_h))$$

and $\max(LB_i + a_j - b_j,\ LB_j)$ in case 1b, can be written as follows:

$$\max(LB_i + a_j - b_j + \sum_{\substack{h \in B_j \cap I_i \\ \& a_h > b_h}} (a_h - b_h),\ LB_j)$$

Thus, LB_{ij} of case 1 can be written as follows:

$$LB_{ij} = \min\{\max(LB_i,\ LB_j + a_i - b_i + \sum_{\substack{h \in B_i \cap I_j \\ \& a_h > b_h}} (a_h - b_h)),$$

$$\max(LB_i + a_j - b_j + \sum_{\substack{h \in B_j \cap I_i \\ \& a_h > b_h}} (a_h - b_h),\ LB_j)\}$$

LB_{ij} of case 2 (Section 9.3.2) can be dealt with in a similar way.

Another obvious way to try to improve the conflict bound is by considering all possible values of i and j (i.e. as given in Section 9.3.2) instead of considering a job i with LB_i=LB.

Finally, it is possible to generalize the conflict bound as follows. Given three unrelated jobs i, j and k (i.e. no arc exists between any two of them), we have the following cases.

Case 1. All three jobs have processing times on the first machine which are larger than their processing times on the second machine. The fact that one of the three jobs is sequenced after the other two, will be used to obtain a lower bound, which is given by:

$$LB_{ijk} = \min[\max(LB_i, LB_j, LB_k + a_i + a_j - b_i - b_j),$$
$$\max(LB_i, LB_k, LB_j + a_i + a_k - b_i - b_k),$$
$$\max(LB_j, LB_k, LB_i + a_j + a_k - b_j - b_k)].$$

Case 2: Two of the three jobs only (jobs i and j, say) have processing times on the first machine which are larger than their processing times on the second machine. As in Case 1, the fact that one of the three jobs is sequenced after the other two will be used to obtain a lower bound, which is given by:

$$LB_{ijk} = \min[\max(LB_i, LB_j, LB_k + a_i + a_j - b_i - b_j),$$
$$\max(LB_i, LB_k, LB_j + a_i - b_i),$$
$$\max(LB_j, LB_k, LB_i + a_j - b_j)].$$

We point out that computing LB_{ijk} in this case will not lead to increasing the lower bound since LB_{ijk}, in this case, is not larger than $\max(LB_k, LB_{ij})$.

Case 3: One of the three jobs only (job i, say) has a processing time on the first machine which is larger than its processing time on the second machine. It appears to be useful, in this case, to use the fact that one of the three jobs is sequenced before the other two to obtain a lower bound, which is given by:

$$LB_{ijk} = \min[\max(LB_i, LB_j, LB_k + b_j - a_j),$$
$$\max(LB_i, LB_k, LB_j + b_k - a_k),$$
$$\max(LB_j, LB_k, LB_i + b_j + b_k - a_j - a_k)].$$

As in case 2, computing LB_{ijk} here will not lead to increasing the lower bound since LB_{ijk}, in this case, is not larger than $\max(LB_i, LB_{jk})$.

Case 4: None of the jobs has a processing time on the first machine which is larger than its processing time on the second machine. As in Case 3, we will use the fact that one of the three jobs is

sequenced before the other two to obtain a lower bound, which is given by:

$$LB_{ijk} = \min[\max(LB_i,\ LB_j,\ LB_k + b_i + b_j - a_i - a_j),\ \max(LB_i,\ LB_k,\ LB_j + b_i + b_k - a_i - a_k),\ \max(LB_j,\ LB_k,\ LB_i + b_j + b_k - a_j - a_k)].$$

Obviously, given k unrelated jobs 1,2,...,k, one can compute a lower bound $LB_{1,2,\ldots,k}$ in a similar way.

CHAPTER TEN

THE GENERAL PERMUTATION FLOW-SHOP PROBLEM UNDER PRECEDENCE CONSTRAINTS

10.1 Introduction

In this chapter we consider the general permutation flow-shop problem under precedence constraints. *The problem can be described as follows. There are n jobs numbered 1,...,n and m machines numbered 1,...,m. Each job i(i=1,...,n) has to be processed on the m machines 1,2,...,m, in that order. The processing time of each job i on each machine k, denoted by* p_{ik}*, is given. Once a job has started on a machine it must be completed on that machine without interruption. The precedence constraints among jobs are represented by a directed acyclic graph G = (V,E), where V denotes the set of vertices and E the set of edges. The vertices of G represent the jobs and the edges represent the arcs between the jobs. Job i must be processed before job j on each machine if there exists a directed path from vertex i to vertex j in E. The objective is to find a sequence of jobs that minimizes the maximum completion time.*

Clearly, this problem is NP-hard since the special case where there are no precedence constraints among jobs, i.e. the $Pm//C_{max}$ problem, is NP-hard (Lenstra, 1977). To the author's knowledge, no one has worked on this problem before.

We shall restrict ourselves to using the definitions and notations used in Chapters 8 and 9. Also, we shall assume that the graph G is made transitive before applying our proposed branch and bound procedure.

Our bounding procedure will be presented in Section 10.2. Section 10.3 contains the full algorithm including further detail about our bounds, branching rule, dominance rules, implementation of the dominance rules and our upper bounding procedure. In Section 10.4 we report on computational experience followed by some concluding remarks in Section 10.5.

10.2 Lower Bound

The lower bound described here is a generalization of the bounds given in Section 8.2.2 for the unconstrained problem.

Given an initial partial sequence σ_1 and a final partial sequence σ_2, we shall derive a lower bound on the maximum completion time for all feasible sequences beginning with the partial sequence σ_1 and ending with the partial sequence σ_2. This is done by relaxing the capacity constraints on some machines, i.e. by allowing some of the machines to process more than one job at the same time. A machine pair (u,v), where $1 \leq u \leq v \leq m$, is chosen and the constraint that machines $u+1,\ldots,v-1$ can process only one job at a time is relaxed. If $u \neq v$, a two-machine sub-problem with precedence constraints is produced in which each job $i \varepsilon S$ (set of unscheduled jobs) has a processing time p_{iu} on the first machine, a processing time p_{iv} on the second machine and a time lag of $\sum_{k=u+1}^{v-1} p_{ik}$ between the completion of processing job i on machine u and the start of processing job i on machine v. (This resulting problem is NP-hard (Monma , —); see also Chapter 9). Using the lower bound derived in Section 9.3.1 for the two-machine problem subject to precedence constraints and the time lag of each job between the two machines u and v, a lower bound, $T(\sigma_1, \sigma_2, u,v)$ for $u \neq v$, for the two machine sub-problem can be written as follows:

$$T(\sigma_1, \sigma_2, u,v) = \max_{j \varepsilon S}\Big(\sum_{i \varepsilon B_j} p_{iu} + p_{ju} + \sum_{k=u+1}^{v-1} p_{jk} + p_{jv} + \sum_{i \varepsilon A_j} p_{iv} + \sum_{i \varepsilon I_j} \min(p_{iu}, p_{iv})\Big) \qquad (10.1)$$

where $B_j = \{h/(h,j) \varepsilon E\}$, $A_j = \{h/(j,h) \varepsilon E\}$ and $I_j = V-(B_j \cup A_j)$

We define $C_1(\sigma_1,k)$ to be the minimum completion time of all jobs sequenced in σ_1 on machine k (if σ_1 is empty we define $C_1(\sigma_1,k)=0$) and

$C_2(\sigma_2,k)$ to be the minimum time between the start of processing jobs in σ_2 on machine k and the completion of processing jobs in σ_2 on the last machine (if σ_2 is empty, we define $C_2(\sigma_2,k)=0$). Now, if we define r_{iu} to be the earliest starting time of job i on machine u and is given by:

$$r_{iu} = \max\{\max_{k=1,\dots,u}\{C_1(\sigma_1,k) + \sum_{h\varepsilon B_i} p_{hk} + \sum_{k'=k}^{u-1} p_{ik'}\}, \max_{h\varepsilon B_i}\{C_1(\sigma_1,1)$$

$$+ \sum_{k=1}^{u} p_{hk} + \sum_{h'\varepsilon B_h} p_{h'1} + \sum_{h'\varepsilon A_h \cap B_i} p_{h'u}$$

$$+ \sum_{h'\varepsilon B_i-(B_h\cup(A_h\cap B_i))} \min\{p_{h'1}, p_{h'u}\}\}\} \qquad (10.2)$$

Where the first term in r_{iu} is a machine based bound based on machine k. It is a generalization of that used by many researchers for the unconstrained permutation flow-shop problem (Ignall & Schrage, 1965; Lomnicki, 1965; Brown & Lomnicki, 1966; McMahon & Burton, 1967; Nabeshima, 1967; Potts, 1974; Bestwick & Hastings, 1976; Lageweg, Lenstra & Rinnooy Kan, 1978; Potts, 1980A). The second term is a job based bound based on jobs in B_i using machines 1 and u.

And define q_{jv} as the minimum time between the completion of job j on machine v and the completion of all jobs and is given by:

$$q_{jv} = \max\{\max_{k=v,\dots,m}\{C_2(\sigma_2,k) + \sum_{h\varepsilon A_j} p_{hk} + \sum_{k'=v+1}^{k} p_{jk'}\}, \max_{h\varepsilon A_j}\{C_2(\sigma_2,m)$$

$$+ \sum_{k=v}^{m} p_{hk} + \sum_{h'\varepsilon B_h \cap A_j} p_{h'v} + \sum_{h'\varepsilon A_h} p_{h'm}$$

$$+ \sum_{h'\varepsilon A_j-(A_h\cup(B_h\cap A_j))} \min\{p_{h'v}, p_{h'm}\}\}\} \qquad (10.3)$$

Due to the symmetry of the problem, the two terms in q_{jv} can be explained in a similar way to that given above.

Then a lower bound, $B(\sigma_1,\sigma_2,u,v)$, for the problem is as follows:

$$B(\sigma_1,\sigma_2,u,v) = \min_{i\epsilon S} r_{iu} + T(\sigma_1,\sigma_2,u,v) + \min_{j\epsilon S} q_{jv} \qquad (10.4)$$

Alternatively, if u=v, a single machine subproblem results in which each job i has a processing time p_{iu}, a release date r_{iu} (calculated using equation 10.2) and a tail q_{iu} (calculated using equation 10.3). This resulting single machine problem with precedence constraints is NP-hard (Lenstra, Rinnooy Kan & Brucker, 1977). Thus a lower bound for this single machine problem is to be used. Such a lower bound can be obtained by setting $q_{iu}=\min_{j\epsilon S} q_{ju}$ for all $i\epsilon S$ and solving the resulting problem using Lawler's algorithm (Lawler, 1973), which sequences a job i with no predecessors and r_{iu} as small as possible first. If $\pi=\{1,\ldots,s\}$, is the sequence obtained using Lawlar's algorithm, then an optimum solution to this problem, and of course a lower bound for original single machine problem is given by:

$$\max_{h=1,\ldots,s} (r_{h,u} + \sum_{i=h}^{s} p_{iu}) + \min_{i=1,\ldots,s} q_{iu} \qquad (10.5)$$

Remark: A different lower bound can be obtained by setting $r_{iu} = \min_{j\epsilon S} r_{ju}$ for all $i\epsilon S$ and solving the resulting problem in a similar way.

A lower bound can also be obtained using a one machine subproblem and disregarding the precedence constraints and solving the resulting problem for which any sequence is optimum. This lower bound is given by:

$$B(\sigma_1,\sigma_2,u,u) = \min_{i\epsilon S} r_{iu} + \sum_{i\epsilon S} p_{iu} + \min_{j\epsilon S} q_{ju} \qquad (10.6)$$

This bound is known as the machine based bound. It is weaker and more quickly computed than 10.4 and 10.5 and will be used to make sure that our bound will not be less than the machine based bound.

More details about our bounds will be given in the following section.

10.3 The Algorithm

10.3.1 Branching Rule

Our branching procedure has the property of adding as few nodes as possible to the search tree. Each node of the search tree corresponds to a job being sequenced either first or last. Let B be the set of jobs with no predecessors and A be the set of jobs with no successors. At every stage, one of the jobs in B is sequenced first if the number of jobs in A is at least equal to the number of jobs in B. Otherwise one of the jobs in A is sequenced last. A formal statement of our branch and bound procedure is given below.

Step 1: Update the two sets B and A. Let n_1 and n_2 be the numbers of jobs in these two sets respectively.

Step 2: If $n_1=1$, sequence the only job in B first and go to Step 5. If $n_2=1$, sequence the only job in A last and go to Step 5.

Step 3: If $n_1 \leq n_2$, sequence a job $i \in B$ with the smallest lower bound (among all jobs in B) first and go to Step 5.

Step 4: ($n_1 > n_2$). Sequence a job $j \in A$ with the smallest lower bound (among all jobs in A) last and proceed to Step 5.

Step 5: Remove the newly sequenced job from V and S. If $S \neq \emptyset$, go to Step 1. Otherwise, evaluate the obtained sequence and use its value as an upper bound on the value of the optimum. Search back for any left nodes (each node corresponds to a job being sequenced first or last) with lower bounds less than the upper bound. If such nodes can be found, update the graph G and the set of unscheduled jobs S and go to Step 1. Otherwise, stop, the procedure has ended.

10.3.2 Lower Bounds

Initial experiments were carried out using the two-machine bound (given by equation 10.4) and the one-machine bound (given by equation 10.5) confirmed the remark by (Potts, 1980A) (for the unconstrained problem) that a two-machine bound is more efficient than a one-machine bound.

The results also showed that calculating r_{iu} and q_{jv} as given in equations 10.2 and 10.3 respectively was too time consuming and that using a weaker but more quickly calculated bound using:

$$\left.\begin{aligned} r_{iu} &= \max_{h=1,\dots u} \left(C_1(\sigma_1,h) + \sum_{k=h}^{u-1} p_{ik}\right) \\ q_{jv} &= \max_{h=v,\dots m} \left(\sum_{k=v+1}^{h} p_{jk} + C_2(\sigma_2,h)\right) \end{aligned}\right\} i,j\varepsilon S \text{ and } u,v=1,\dots,m \qquad (10.7)$$

gave good computational results.

Having decided to use the lower bounds $B(\sigma_1,\sigma_2,u,v)$ given by 10.4 and 10.6 (r_{iu} and q_{jv} as in 10.7), the choice of machine pairs is discussed next.

In (Lageweg et al., 1978) and (Potts, 1974) it was found that the sets of machine pairs $\{(1,m),\dots,(m-1,m)\}$ and $\{(1,m),\dots,(m,m)\}$ respectively gave good computational results for the unconstrained problem. Finally, it was found in (Potts, 1980A) that the set of machine pairs $\{(1,1),\dots,(m,m),(1,m),\dots,(m-1,m)\}$ gave good computational results.

With all this in mind we have decided to use the set of machine pairs $\{(1,m),\dots,(m-1,m)\}$. To ensure that our proposed bound is never less than the machine-based bound, the set of machine pairs $\{(1,1,),\dots,(m,m)\}$ will also be used. We conclude that the set of machine pairs to be used is given as follows:

$$W = \{(1,1),\dots,(m,m),(1,m),\dots,(m-1,m)\} \qquad (10.8)$$

Thus an overall lower bound, $LB(\sigma_1,\sigma_2,W)$ for the problem is given by:

$$LB(\sigma_1,\sigma_2,W) = \max(B(\sigma_1,\sigma_2,u,v)/(u,v)\varepsilon W) \qquad (10.9)$$

10.3.3 Dominance Rules

In this section we shall be interested in finding conditions under which a particular node can be eliminated before its lower bound is calculated.

Dominance rules are particularly useful when a node with a lower bound which is less than the optimum, can be eliminated.

Using the notations of the previous sections, let $i,j \in B$. We now define:

$$\Delta_{1k} = C_1(\sigma_1 ij,k) - C_1(\sigma_1 j,k), \quad k=1,\dots,m$$

Also, for two jobs $i', j' \varepsilon A$, we define:

$$\Delta_{2k} = C_2(j'i'\sigma_2,k) - C_2(j'\sigma_2,k), \quad k=1,\dots,m$$

Theorem 10.1

If for two jobs i and $j \varepsilon B$:

$$\Delta_{1k-1} \leqslant \Delta_{1k} \leqslant p_{ik}, \quad for\ k=2,\dots,m \tag{10.10}$$

then $\sigma_1 ij$ dominates $\sigma_1 j$.

Proof

Let $\pi = (\pi(1),\pi(2),\dots,\pi(r))$ be an arbitrary sequence of jobs such that:

$$\pi \subset S - \{i,j\}$$

and that $\sigma_1 j\pi i$ is a feasible partial sequence.

The proof continues as given in (Szwarc, 1971) for the unconstrained problem. We shall prove that:

$$(10.10) \text{ implies } C_1(\sigma_1 ij\pi,k) - C_1(\sigma_1 j\pi,k) \leqslant C_1(\sigma_1 ij,k) - C_1(\sigma j,k)$$

for all k 1,...,m.

For $\pi=\emptyset$, (10.10) is trivially true. For $\pi\neq\emptyset$, we have the following:

Step 1: Let r=1, then $\pi=\pi(1)$. Proof by induction. The theorem is true for k=1 since $C_1(\sigma_1 ij\pi(1),1) - C_1(\sigma_1 j\pi(1),1) = p_{i1} = C_1(\sigma_1 ij,1) - C_1(\sigma_1 j,1)$. Suppose the theorem is true for k=h-1. We will prove it for k=h. Consider

$$C_1(\sigma_1\ ij\pi(1),h) - C_1(\sigma_1 j\pi(1),h) = \max\{C_1(\sigma_1 ij\pi(1),h-1),$$

$$C_1(\sigma_1 ij,h)\} + p_{\pi(1)h} - \max\{C_1(\sigma_1 j\pi(1),h-1),\ C_1(\sigma_1 j,h)\} - p_{\pi(1)h}$$

$$\leq \max\{C_1(\sigma_1 ij\pi(1),h-1) - C_1(\sigma_1 j\pi(1),h-1),\ C_1(\sigma_1 ij,h) - C_1(\sigma_1 j,h)\}$$

$$\leq \max\{C_1(\sigma_1 ij,h-1) - C_1(\sigma_1 j,h-1),\ C_1(\sigma_1 ij,h) - C_1(\sigma_1 j,h)\}$$

$$\leq C_1(\sigma_1 ij,h) - C_1(\sigma_1 j,h) = \Delta_k$$

This concludes the proof for $r=1$.

Step 2: Let $r=2$, then $\pi=(\pi(1),\pi(2))$. For the case when $k=1$, the theorem holds. Assuming the theorem is true for $k=h-1$, we will prove it for $k=h$. As in Step 1 we have:

$$C_1(\sigma_1 ij\pi(1)\pi(2),h) - C_1(\sigma_1 j\pi(1)\pi(2),h) \leq \max\{C_1(\sigma_1 ij\pi(1)\pi(2),h-1)$$

$$- C_1(\sigma_1 j\pi(1)\pi(2),h-1),\ C_1(\sigma_1 ij\pi(1),h) - C_1(\sigma_1 j\pi(1),h)\}$$

$$\leq \max\{C_1(\sigma_1\ ij,h-1) - C_1(\sigma_1 j,h-1),\ C_1(\sigma_1 ij\pi(1),h) - C_1(\sigma_1 j\pi(1),h)\}$$

$$\leq \max\{C_1(\sigma_1 ij,h-1) - C_1(\sigma_1 j,h-1),\ C_1(\sigma_1 ij,h) - C_1(\sigma_1 j,h)\}$$

$$\leq C_1(\sigma_1 ij,h) - C_1(\sigma j,h)$$

By performing Steps 3, 4, ..., r one can prove the necessary result.

Corollary 10.1

If for two jobs i' and $j'\varepsilon A$:

$$\Delta_{2k} \leq \Delta_{2\ k-1} \leq p_{i'k-1}, \quad \textit{for } k=2,\ldots,m \qquad (10.11)$$

then $j'i'\sigma_2$ dominates $j'\sigma_2$.

Proof

Similar to the proof of the theorem above.

10.3.4 Implementation of the Dominance Rules

The dominance rules from Theorem 10.1 and Corollary 10.1 are checked at every node of the search tree (i.e. after performing Steps 1 and 2 and before performing Step 3 of the branching procedure of Section 10.3.1).

It is clear that for condition 10.10 to hold we must have

$$p_{i1} \leqslant p_{ik} \quad \text{for } k=2,\ldots,m \tag{10.12}$$

Also, for condition (10.11) to hold we must have

$$p_{i'm} \leqslant p_{i'k} \quad \text{for } k=1,\ldots,m-1 \tag{10.13}$$

Conditions (10.10) and (10.11) need not be transitive. For this reason we have to check these two conditions for each pair (i,j) such that $i,j \in B$ and that (10.12) holds for job i in the first case, and for each pair (i',j') such that $i',j' \in A$ and that (10.13) holds for job i' in the second case.

If $n_2 < n_1$ (using the notations of Section 10.3.1), we start by checking condition (10.11) for each pair of jobs (i',j'), $i',j' \in A$ to eliminate as many nodes as possible from being candidates for the last available position. If only one job is left as a candidate for the last position then this job is sequenced last; otherwise we check condition (10.10) for each pair of jobs (i,j), $i,j \in B$ to eliminate as many nodes as possible from being candidates for the first available position. If only one job is left as a candidate for the first position, then this job is sequenced first; otherwise we proceed to Step 3 of Section 10.3.1. The case when $n_1 \leqslant n_2$ is dealt with in a similar way except that here we start by checking condition (10.10) and then (if necessary) we check condition (10.11). The application of conditions (10.10) and conditions (10.11) require $O(m\, n_1^2)$ and $O(mn_2^2)$ steps respectively.

Let n_1' and n_2' be the number of candidates for the first and last available positions after applying conditions (10.10) and (10.11)

respectively (it is clear that $n_1' \leq n_1$ and $n_2' \leq n_2$).

Initial experiments showed that replacing n_1 and n_2 in Steps 3 and 4 of Section 10.3.1 by n_1' and n_2' respectively, led to a slightly worse result. For this reason, the idea of using dominance rules to direct our branching procedure will be abandoned.

10.3.5 *Upper Bounds*

It is well known that computation can be reduced by using a heuristic to find a good solution to act as an upper bound on the maximum completion time before the start of the branch and bound procedure.

As stated before, whenever a complete sequence of scheduled jobs is obtained using the branch and bound algorithm, the maximum completion time of the jobs ordered in this sequence is calculated and used as an upper bound on the maximum completion time. Additionally, a heuristic method is used once at the top of the tree to obtain an initial upper bound. This heuristic is as follows. Firstly, disregarding the precedence constraints, we apply Campbell's method (Campbell et al., 1970; also given in Section 8.2.4) which requires applying Johnson's $F2//C_{max}$ algorithm using processing times for job i $(i=1,\ldots,n)$ of $\sum_{k=1}^{h} p_{ik}$ and $\sum_{k=m+1-h}^{m} p_{ik}$ to obtain a sequence. This sequence is then evaluated as an $Pm//C_{max}$ schedule. This procedure is applied for values $h=1,\ldots,m-1$. Let $\pi'=(\pi'(1),\pi'(2),\ldots,\pi'(n))$ be the best sequence obtained. Secondly, we reorder jobs in π' to form a new sequence π as follows. Each step will sequence a job $\pi'(i)$ with no predecessors and i as small as possible first. This procedure is repeated until all jobs have been assigned positions in π. The sequence π is then evaluated as a $Pm/prec/C_{max}$ schedule. The maximum completion time of jobs sequenced in π is used as an upper bound on the value of C_{max}. This procedure requires $O(\max\{mn \log n, n^2\})$ steps.

10.4 Computational Experience

10.4.1 Test Problems

The algorithms were tested on problems of sizes up to 40 jobs. These test problems consisted of problems with random processing times (R), problems with correlation between the processing times of each job (C), problems for which the random processing times of each job have a positive (T^+) or negative (T^-) trend, and finally, problems with correlation and a positive (CT^+) or a negative (CT^-) trend for the processing times of each job.

For each test problem with n jobs and m machines, mn integer data p_{ik} were generated from uniform distributions $[\alpha_{ik},\beta_{ik}]$. For problems with correlation, n additional integers θ_i were randomly chosen from {1,2,3,4,5}. Problems with negative trends were obtained by renumbering machine k as m-k+1 for k=1,...,m. Values of α_{ik} and β_{ik} for different classes of test problems are given in Table 10.1.

This method of processing times generation follows that of Lageweg, Lenstra and Rinnooy Kan (Lageweg et a., 1978).

Table 10.1: Test Data

$\alpha_{ik} : \beta_{ik}$	Random	Correlated
No trend	$1 : 100$	$20\theta_i+1 : 20\theta_i+20$
Positive trend	$12\frac{1}{2}(k-1)+1 : 12\frac{1}{2}(k-1)+100$	$2\frac{1}{2}(k-1)+20\theta_i+1 : 2\frac{1}{2}(k-1)+20\theta_i+20$

In the precedence graph G, each arc (i,j) with i<j was included with a given probability p. The following values of p have been considered: 0.0, 0.1, 0.2, 0.3, 0.4, 0.5 and 0.75.

For each set of values (p,n,m), 40 problems were generated from the six different classes of problems according to Table 10.2.

Table 10.2: Number of Test Problems for each set of values (p,n,m)

Problem Class	Number of Test Problems
Random (R)	10
Correlated (C)	10
Random with positive trend (T^+)	5
Random with negative trend (T^-)	5
Correlated with positive trend (T^+)	5
Correlated with negative trend (T^-)	5

10.4.2 Computational Results

The algorithms were coded in FORTRAN IV and run on a CDC 7600 computer.

Computational results are given in Tables 10.3, 10.4 and A.1.1. Whenever a problem was not solved after 50,000 nodes had been generated, computation was abandoned for that problem. Thus, in some cases, the figures shown in Table 10.3 will be lower bounds on the average computation times (A.C.T.) or lower bounds on the average number of nodes (A.N.N.).

Average computation times and average number of nodes for our proposed lower bound are given in the first two columns of Table 10.3. Adding our dominance rules and upper bounding procedure, the corresponding results are given in the third and fourth columns of the same table. Numbers of unsolved problems classified according to the value of p(p=0.1, 0.2, 0.3, 0.4, 0.5, 0.75) and according to problem class (R, C, T^+, T^-, CT^+, CT^-) are given in Table 10.4. The precise numbers of unsolved problems for each set of values (p,n,m), classified according to problem class, are given in Table A.1.1.

Given a particular value of p, columns 3 and 4 of Table 10.3 and column 2 of Table 10.4 show that using dominance rules and the heuristic reduces computation and increases the efficiency of the algorithm. However, the major increase in the efficiency of the algorithm is due to the dominance rules, which is consistent with the results of references (Lageweg, Lenstra & Rinnooy Kan, 1978; Potts, 1980A) for the unconstrained permutation flow-shop problem.

As expected, increasing the value of p (for particular values of n and m) decreases average computation time, average number of nodes and number of unsolved problems. An unexpected result is observed in the second column of Table A.1.1, where increasing the value of p from 0.0 to 0.1 (for n=20 and m=3) led to an increase in the number of unsolved problems from 13 to 18. This was due to the effect of using the dominance rules and the heuristic on the (0.0,20,3) problems.

Using the first column of Table 10.4, Table 10.5 shows the order of the different classes of problems for the different values of p obtained according to the percentage of the unsolved problems to the total number of test problems from each of these classes (see footnote of Table 10.4). The first and last rows of Table 10.5 contain the hardest and easiest classes of problems for the different values of p respectively.

Table 10.3: Average Computation Time and Average Number of Nodes*

p	n	m	LB A.C.T.*	LB A.N.N.*	LB+D+H A.C.T.*	LB+D+H A.N.N.*
0.0	8	5	0.10	532	0.03	134
	8	7	0.20	812	0.17	640
	10	3	0.89	8,879	0.39	3,236
	10	5	1.23	6,836	1.04	5,284
	10	7	2.11	8,723	1.62	6,115
	15	3	4.47	21,553	1.20	5,739
	15	5	7.74	28,398	7.22	25,523
	20	3	10.64	31,487	5.81	19,621
	20	5	-	-	-	-
	30	3	-	-	-	-
	40	3	-	-	-	-
0.1	8	5	0.06	290	0.04	168
	8	7	0.09	304	0.07	246
	10	3	0.38	3,738	0.14	1,002
	10	5	0.74	3,964	0.72	3,381
	10	7	0.89	3,504	0.47	1,630
	15	3	3.28	16,940	1.16	5,621
	15	5	8.35	27,961	8.37	26,112
	20	3	7.14	32,398	5.50	25,058
	20	5	-	-	-	-
	30	3	-	-	-	-
	40	3	-	-	-	-
0.2	8	5	0.03	103	0.02	73
	8	7	0.06	199	0.05	138
	10	3	0.05	317	0.02	63
	10	5	0.22	1,203	0.17	786
	10	7	0.52	1,977	0.41	1,394
	15	3	1.95	10,576	0.90	4,555
	15	5	3.79	13,944	3.23	10,741
	20	3	6.21	26,018	3.89	16,125
	20	5	13.27	30,624	12.90	28,822
	30	3	9.21	22,897	9.18	20,573
	40	3	12.40	26,707	14.83	25,799
0.3	8	5	0.02	65	0.01	43
	8	7	0.03	89	0.02	63
	10	3	0.02	110	0.01	43
	10	5	0.06	295	0.04	115
	10	7	0.14	445	0.12	367
	15	3	0.97	6,420	0.34	1,874
	15	5	1.37	4,493	0.04	115
	20	3	3.98	16,745	1.04	4,356
	20	5	7.38	18,529	6.60	16,229
	30	3	10.33	18,026	6.17	12,988
	40	3	13.68	22,780	13.51	19,060

p	n	m	LB A.C.T.*	LB A.N.N.*	LB+D+H A.C.T.*	LB+D+H A.N.N.*
			Algorithm			
0.4	8	5	0.01	34	0.01	22
	8	7	0.02	45	0.02	37
	10	3	0.01	37	0.01	18
	10	5	0.03	126	0.02	72
	10	7	0.04	97	0.04	79
	15	3	0.17	1,074	0.08	361
	15	5	0.30	1,011	0.21	652
	20	3	2.07	9,445	0.72	3,642
	20	5	4.52	13,583	4.12	11,368
	30	3	4.81	12,287	3.99	9,454
	40	3	14.39	18,292	11.29	17,519
0.5	8	5	0.01	22	0.01	14
	8	7	0.01	28	0.01	22
	10	3	0.01	42	0.01	15
	10	5	0.02	52	0.02	35
	10	7	0.02	35	0.02	26
	15	3	0.07	348	0.05	154
	15	5	0.12	354	0.11	304
	20	3	1.06	5,535	0.26	903
	20	5	2.72	8,559	1.99	5,694
	30	3	3.92	8,754	2.24	4,422
	40	3	8.35	12,845	6.99	10,137
0.75	8	5	0.00	7	0.00	5
	8	7	0.01	8	0.01	6
	10	3	0.01	10	0.01	4
	10	5	0.01	8	0.01	4
	10	7	0.01	10	0.01	8
	15	3	0.02	12	0.02	5
	15	5	0.03	16	0.03	13
	20	3	0.07	70	0.06	20
	20	5	0.15	355	0.13	210
	30	3	0.38	318	0.27	48
	40	3	2.00	1,289	0.90	176

* - Lower bound on the average when there are unsolved problems.

- Times are in CPU seconds.

- A.C.T. Average Computation Time
- A.N.N. Average Number of Nodes
- LB Lower Bound
  D Dominance Rules
  H Heuristic
  \- Most problems were left unsolved

Table 10.4: Numbers of Unsolved Problems for Different Values of p*

p	Problem Class	Algorithm LB	LB+D+H
0.0	R	44	41
	C	52	37
	T^+	19	15
	T^-	27	23
	CT^+	29	22
	CT^-	26	21
0.1	R	37	37
	C	49	41
	T^+	19	18
	T^-	20	19
	CT^+	27	23
	CT^-	25	20
0.2	R	19	17
	C	38	30
	T^+	4	3
	T^-	4	4
	CT^+	15	9
	CT^-	14	9
0.3	R	13	11
	C	24	18
	T^+	0	0
	T^-	7	3
	CT^+	6	2
	CT^-	6	4
0.4	R	7	6
	C	19	14
	T^+	1	1
	T^-	3	3
	CT^+	1	1
	CT^-	4	3
0.5	R	6	4
	C	11	6
	T^+	1	0
	T^-	1	1
	CT^+	2	1
	CT^-	0	0

* All problems with p = 0.75 were solved.

- For every value of p (p=0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.75), 110 problems were generated from each of the two classes R and C and 55 problems were generated from each of the other classes of problems.

Table 10.5

p	0.0	0.1	0.2	0.3	0.4	0.5
Hardest	CT^+	CT^+	C	C	C	C
	T^-	CT^-	CT^+	T^-	CT^-	R
	C	C	CT^-	R	R	CT^+
	CT^-	T^-	R	CT^-	T^-	T^-
	R	T^+	T^-	CT^+	CT^+	T^+
Easiest	T^+	R	T^+	T^+	T^+	CT^-

The first row of Table 10.5 shows that the problems with correlation and a positive trend for the processing times of each job (CT^+) to be the hardest for p=0.0 and 0.1 while problems with correlation between the processing times of each job (C) to be the hardest for all other values of p. The last row of the same table shows problems with positive trends for the processing times of each job to be the easiest in most cases.

However, if we consider the percentage of the number of all unsolved problems to the total number of test problems (770 problems of each of the two classes R and C and 385 problems of each of the other classes were tested), one can order the different classes as follows: C, CT^+, CT^-, R, T^-, T^+, where problems in class C are the hardest and problems in class T^+ are the easiest. Thus, the correlated problems appear to be the most challenging, which is consistent with the findings of references (Lageweg, Lenstra & Rinnooy Kan, 1978; Potts, 1980A) for the unconstrained case and with that of Chapter 9 for the two-machine flow-shop problem under precedence constraints.

Finally, using Table 10.4, one can order the different classes of problems according to the effect of the dominance rules on these different classes for the different values of p. These orders are given in Table 10.6. The effectiveness of the dominance rules on each class was measured by the percentage of the reduction in the number of unsolved problems when using these dominance rules to the number of unsolved problems when the lower bound only is used. The first and last rows of Table 10.6 contain the most and least affected classes of problems for the different values of p respectively.

Table 10.6

p	0.0	0.1	0.2	0.3	0.4	0.5
Most affected	C	CT^-	CT^+	CT^+	C	T^+
	CT^+	C	CT^-	T^-	CT^-	CT^+
	T^+	CT^+	T^+	CT^-	R	C
	CT^-	T^+	C	C	CT^+	R
	T^-	T^-	R	R	T^+	CT^-
Least affected	R	R	T^-	T^+	T^-	T^-

Table 10.6 shows that the dominance rules to be most effective on the problem classes C, CT^+ and CT^- and most ineffective on the problem classes R and T^-.

Table A.1.1 shows that most of these reduction in the number of unsolved problems occurred when m=3. However, for m≥5, the dominance rules were most effective when p=0.0.

10.5 Concluding Remarks

The branch and bound procedure proposed in this chapter forms the first work that has been done on the permutation flow-shop problem under precedence constraints.

This branch and bound procedure enabled us to solve problems with up to 40 jobs. The computational results showed that the performance of the dominance rules was remarkably good. As expected, the efficiency of the algorithm increases as the value of p increases. It was most ineffective when p=0.0. Obviously, in this case (p=0.0), one would choose to use the branch and bound procedure proposed by Potts (Potts, 1980A; see also Chapter 8). However, even when p=0.1 or 0.2, our branch and bound algorithm performed badly, especially on correlated problems. This indicates that a different approach is needed for these cases. An approach based on selecting certain pairs of jobs i and j (where no arc joining jobs i and j exists in E) and deciding, at the top of the search tree, an ordering between the two jobs of each pair seems worth investigating. One way of selecting these pairs of jobs would be by selecting a pair of jobs i and j (where arcs (i,j) and (j,i) are not in E) such that job i has the largest number of in-going arcs and job j has the largest number of out-going arcs.

Improving the two-machine lower bound, perhaps as given in Chapter 9, should also yield a more efficient algorithm.

PART IV

CONCLUSION

CHAPTER ELEVEN

CONCLUSION

11.1 Contribution of this Research

As mentioned in Chapter 1, this thesis is devoted to scheduling problems. Emphasis has been on deriving optimal branch and bound algorithms for two single-machine problems and two multi-machine problems. We have also given some attention to heuristic methods (methods which do not guarantee optimal solutions) because these methods dominate all other methods in real life situations.

We started Chapter 5 by giving a review of one machine heuristics. We then completed our comprehensive list of one machine heuristics by suggesting four other one machine heuristics. A tree type heuristic was also included. The basis of this heuristic is to select one node at each level of the tree from which to branch. This node is selected using one of the following two methods: (a) it has the smallest lower bound in which case the tree type heuristic is referred to by H_L; (b) it has the smallest value of a second order heuristic H, in which case the tree type heuristic is referred to by H_H. The performance of one of the proposed heuristics together with heuristics H_L and H_H was tested on the $1/r_i/\Sigma w_i C_i$ problem. The test problems included problems with up to 50 jobs. Optimal or sub-optimal solutions to these problems were used to compare the performance of the heuristics. The results showed heuristic H_L to be substantially better than heuristic H_H. They also indicated that applying both heuristics (i.e. H_L and H_H) and choosing the best solution is a reasonable strategy. It would be interesting to see how heuristics H_L and H_H together with other proposed heuristics perform on different problems and also on problems of sizes larger than 50.

In Chapter 6 we proposed branch and bound algorithms to solve the $1/r_i/\Sigma w_i C_i$ problem. Problems with up to 50 jobs were used to test the performance of our proposed algorithms. The results showed all our algorithms to work reasonably well, especially problems with up to 40 jobs.

In Chapter 7 we proposed a branch and bound algorithm for the $1//\Sigma w_i C_i^2$ problem. Problems with up to 70 jobs were used to test the performance of our proposed algorithm. The excellent results we had were not expected: all the 700 test problems but one were solved without the need for branching. We have also explained how our proposed bounding procedure can be applied to the $1/prec/\Sigma w_i C_i^2$ problem. It would be interesting to test the performance of our proposed bounding procedure in a branch and bound algorithm for the $1/prec/\Sigma w_i C_i^2$ problem.

Branch and bound algorithms for the $F2/prec/C_{max}$ problems are given in Chapter 9. The performance of our algorithms was assessed using test problems with up to 60 jobs. All problems with up to 40 jobs and most problems with 50 jobs were solved using our best algorithm. Unfortunately, our algorithms were not so effective on problems with 60 jobs. Methods of improving our bounding procedure to deal with this case were also included.

Finally, in Chapter 10 we gave a branch and bound algorithm to solve the $Pm/prec/C_{max}$ problem. To the author's knowledge no-one has worked on this problem before. The test problems included problems with n/m: 8/5, 8/7, 10/3, 10/5, 10/7, 15/3, 15/5, 20/3, 20/5, 30/3, 40/3. The results showed our proposed algorithm to work reasonably well, especially on problems with large values of p. Including dominance rules in the algorithm lead to remarkably better results.

11.2 Future of Scheduling

With the continuing dramatic reduction in the size and cost of computer hardware, it is becoming a viable proposition to build computers

with thousands of individual processors suitably linked together. Each one of these processors is capable of executing a non-trivial program. The assumption that as many processors as needed are available has been made in almost all work done on parallel computing. Although this assumption is unrealistic, a parallel algorithm (an algorithm which is adapted for parallel computers) will in practise be run on a machine with a finite number of processors. (The complexity of a parallel algorithm depends very much on the structure of the parallel computer on which it is run.)

Several parallel models have been proposed and studied by researchers. Two important models are the *Single Instruction, Multiple Data stream (SIMD) model* and the *Multiple Instruction Multiple Data stream (MIMD) model.*

SIMD computers are characterized by the following (Dekel & Sahni, 1980):

1. They consist of M processing elements (PEs) indexed 0,1,...,M-1. Each element (PE) knows its index and is capable of performing the standard arithmetic and logical operations.
2. Each PE has a local memory.
3. The PEs operate simultaneously and under the control of a single instruction stream. (This means all the PEs execute the same program simultaneously.)
4. A subset of the PEs may be chosen to perform an instruction. The remaining PEs will be left idle.

The MIMD computers are also characterized by the above four points except that point 3 is replaced by:

3. Each PE may operate independent of all other PEs. (This means the PEs do not have to operate simultaneously nor under a single instruction stream.)

Dekel and Sahni (Dekel & Sahni, 1980) gave (among other things) $O(\log^2 n)$ parallel algorithms (based on computation trees) to solve the following scheduling problems:

(a) Scheduling n jobs on one machine to minimize the maximum lateness. Pre-emptions are permitted.

(b) Scheduling n jobs on one machine to minimize the number of late jobs.

(c) Scheduling n jobs on one machine to minimize the sum of completion times subject to deadlines.

The complexity of the fastest sequential algorithm known for each of the above problems is $0(n \log n)$.

If A is a parallel algorithm for a problem P, the *effectiveness of processor utilization* (EPU) is defined as follows:

$$EPU(P,A) = \frac{\text{Complexity of the fastest sequential algorithm for P}}{\text{Number of PEs used by A} \times \text{Complexity of A}}$$

Each parallel algorithm for each of the above scheduling problems uses $0(n)$ PEs and thus has $EPU(P,A) = 0(n \log n/(n \log^2 n)) = 0\ (1/\log n)$. We point out that the best EPU one can hope for is $0(1)$. Some parallel algorithms that achieve this EPU can be found in (Dekel & Sahni, 1980).

Of the SIMD models, Dekel and Sahni (Dekel & Sahni, ——) considered a model with only the shared memory (SMM), i.e. a model with a large common memory which is shared by all PEs. In their model it is assumed that any PE can access any word of the common memory in $0(1)$ time and that not more than one PE can access to read from or write in the same word simultaneously. They gave (among other things) parallel algorithms to solve the following scheduling problems:

(a) Scheduling n jobs with unit processing times on one machine to minimize the number of late jobs.

(b) Scheduling n jobs with unit processing times on one machine to minimize the sum of completion times subject to deadlines.

Each of the parallel algorithms requires $0(\log n)$ time, uses $0(n^2)$ PEs and has EPU that is of $0(1/n)$. We remark that the fastest sequential

algorithm known for each of the above problems requires $O(n \log n)$ time.

Clearly, given an MIMD computer with a large number of processing elements (PEs) suitably linked together, it should be possible to design branch and bound algorithms for solving NP-hard problems of greater size than has so far been possible. Of course, if every alternative at each stage of a branch and bound algorithm is pursued in parallel then this algorithm executes in polynomial time. The problem that may arise here is that unless each PE is directly connected to each of the other PEs, not all PEs can, in general, be made immediately aware of the new best value of the upper bound. Unfortunately, a model that can overcome this problem is not appropriate because of the large number of the PEs involved. However, a suitable choice of network (model) is one in which each PE is connected to just a small number of topologically neighbouring PEs. The r-ary n-cube network considered in (Burton et al., 1981) conforms to this requirement.

The r-ary n-cube network consists of nr^n PEs, each of which has its own memory and with a sub-section for handling communication. Each PE in this network contains the same program and enjoys two-way communication with exactly 2 r selected neighbouring PEs. Any idle PE repeatedly requests work (if any) from and exchanges information with each of the PEs to which it is connected. Each PE will apply branching rules, elimination rules, a lower bound and an upper bound locally to all sub-problems it is going to work on. For further detail about this r-ary n-cube network, we refer to the above given reference.

APPENDIX 1

Table A.1.1: Numbers of Unsolved Problems*

p	n	m	Problem Class	LB	LB+D+H
0.0	10	3	R	1	0
			C	1	0
			T^+	0	0
			T^-	2	1
			CT^+	1	0
			CT^-	0	0
0.0	10	5	R	1	1
			C	0	0
			T^+	0	0
			T^-	1	1
			CT^+	1	0
			CT^-	0	0
0.0	10	7	R	0	0
			C	1	0
			T^+	0	0
			T^-	2	2
			CT^+	1	0
			CT^-	0	0
0.0	15	3	R	2	1
			C	6	0
			T^+	0	0
			T^-	2	0
			CT^+	2	1
			CT^-	3	1
0.0	15	5	R	5	5
			C	6	3
			T^+	4	0
			T^-	3	3
			CT^+	4	4
			CT^-	3	3
0.0	20	3	R	5	4
			C	8	4
			T^+	0	0
			T^-	2	1
			CT^+	5	2
			CT^-	5	2

p	n	m	Problem Class	LB	LB+D+H
0.1	10	3	R	0	0
			C	0	0
			T^+	0	0
			T^-	1	0
			CT^+	1	0
			CT^-	0	0
0.1	10	5	R	0	0
			C	0	0
			T^+	0	0
			T^-	1	1
			CT^+	0	0
			CT^-	0	0
0.1	10	7	R	0	0
			C	1	0
			T^+	0	0
			T^-	0	0
			CT^+	0	0
			CT^-	0	0
0.1	15	3	R	1	1
			C	2	1
			T^+	1	0
			T^-	0	0
			CT^+	3	0
			CT^-	3	1
0.1	15	5	R	4	4
			C	7	6
			T^+	0	0
			T^-	1	1
			CT^+	4	4
			CT^-	2	1
0.1	20	3	R	2	2
			C	9	4
			T^+	3	3
			T^-	2	2
			CT^+	4	4
			CT^-	5	3
0.2	15	3	R	1	1
			C	3	1
			T^+	0	0
			T^-	0	0
			CT^+	3	1
			CT^-	0	0

p	n	m	Problem Class	LB	LB+D+H
0.2	15	5	R	2	2
			C	3	1
			T^+	0	0
			T^-	0	0
			CT^+	1	1
			CT^-	1	1
0.2	20	3	R	3	2
			C	7	3
			T^+	1	1
			T^-	0	0
			CT^+	4	2
			CT^-	4	2
0.2	20	5	R	5	5
			C	9	9
			T^+	2	2
			T^-	1	1
			CT^+	4	1
			CT^-	3	1
0.2	30	3	R	1	1
			C	8	8
			T^+	1	0
			T^-	2	2
			CT^+	3	2
			CT^-	3	3
0.2	40	3	R	7	6
			C	8	8
			T^+	0	0
			T^-	1	1
			CT^+	2	2
			CT^-	3	2
0.3	15	3	R	2	1
			C	0	0
			T^+	0	0
			T^-	0	0
			CT^+	1	0
			CT^-	0	0
0.3	15	5	R	0	0
			C	0	0
			T^+	0	0
			T^-	0	0
			CT^+	0	0
			CT^-	0	0

p	n	m	Problem Class	LB	LB+D+H
0.3	20	3	R	2	1
			C	4	1
			T^+	0	0
			T^-	3	0
			CT^+	1	0
			CT^-	1	0
0.3	20	5	R	2	2
			C	6	6
			T^+	0	0
			T^-	2	2
			CT^+	1	1
			CT^-	1	0
0.3	30	3	R	2	2
			C	6	4
			T^+	0	0
			T^-	1	1
			CT^+	1	0
			CT^-	2	2
0.3	40	3	R	5	5
			C	8	7
			T^+	0	0
			T^-	1	0
			CT^+	2	1
			CT^-	2	2
0.4	20	3	R	1	0
			C	2	0
			T^+	0	0
			T^-	0	0
			CT^+	0	0
			CT^-	1	1
0.4	20	5	R	0	0
			C	6	6
			T^+	0	0
			T^-	1	1
			CT^+	0	0
			CT^-	1	0
0.4	30	3	R	1	1
			C	6	3
			T^+	0	0
			T^-	1	1
			CT^+	0	0
			CT^-	1	1

p			Problem Class	LB	LB+D+H
0.4	40	3	R	5	5
			C	5	5
			T^+	1	1
			T^-	1	1
			CT^+	1	1
			CT^-	1	1
0.5	20	3	R	0	0
			C	1	0
			T^+	0	0
			T^-	0	0
			CT^+	1	0
			CT^-	0	0
0.5	20	5	R	0	0
			C	4	3
			T^+	0	0
			T^-	0	0
			CT^+	0	0
			CT^-	0	0
0.5	30	3	R	1	0
			C	4	1
			T^+	0	0
			T^-	0	0
			CT^+	0	0
			CT^-	0	0
0.5	40	3	R	5	4
			C	2	2
			T^+	1	0
			T^-	1	1
			CT^+	1	1
			CT^-	0	0

* All problems with p=0.75 were solved.

- For p=0.0 and 0.1, most problems with n/m=20/5, 30/3 and 40/3 were left unsolved.

REFERENCES

ADOLPHSON, D. L. & HU, T. C. Optimal linear ordering. *SIAM J. Appl.Math.* 25 (1973), 403-423.

ASHOUR, S. Sequencing Theory. *Springer Verlag* (1972).

ASHOUR, S. & QURAISHI, M. N. Investigation of various bounding procedures for production scheduling problems. *Internat. J. Production Res.*, 7 (1969), 249-252.

BAGGA, P. C. & CHAKRAVARTI, N. K. Optimal m-stage production schedules. *J. of Canadian Operations Res. Society*, 6 (1968), 71-78.

BAGGA, P. C. & KALRA, K. R. A node elimination procedure for Townsend's Algorithm for solving the single machine quadratic penalty function scheduling problem. *Management Science*, 26 (1980), 633-636.

BAKER, K. R. Introduction to sequencing and scheduling. *John Wiley & Sons*, (1974).

BAKER, K. R. A comparative study of flow-shop algorithms. *Oper. Res.*, 23 (1975), 62-73.

BAKER, K. R. Computational experience with a sequencing algorithm adapted to the tardiness problem. *AIIE Transactions*, 9 (1977), 32-35.

BAKER, K. R. & SCHRAGE, L. E. Finding an optimal sequence by dynamic programming. An extension to precedence-related tasks. *Oper. Res.*, 26, No. 1 (1978A).

BAKER, K. R. & SCHRAGE, L. E. Dynamic programming solution of sequencing problems with precedence constraints. *Oper. Res.*, 26, No.3 (1978B).

BAKER, K. R. & SU, Z. S. Sequencing with due-dates and early start times to minimize maximum tardiness. *Naval Research Logistics Quarterly*, 21 (1974), 171-176.

BALAS, E. & CHRISTOFIDES, N. A restricted lagrangean approach to the travelling salesman problem. *Math. Programming*, 21 (1981), No.1, 19-46.

BANSAL, S. P. Single machine scheduling to minimize weighted sum of completion times with secondary criterion: A branch and bound approach. *European Journal of Oper. Res.*, 5 (1980), 177-181.

BARNES, J. W. & BRENNAN, J. J. An improved algorithm for scheduling jobs on identical machines. *AIIE Transactions*, 9, No. 1 (1977).

BESTWICK, P. F. & HASTINGS, N. A. J. A new bound for machine scheduling. *Oper. Res. Quart.*, 27 (1976).

BOWMAN, E. H. The schedule-sequencing problem. *Oper. Res.*, 7 (1959), 621-624.

BROWN, A. P. G. & LOMNICKI, Z. A. Some applications of the branch and bound algorithm to machine scheduling problems. *Oper. Res. Quart.*, 2 (1966), 173-186.

BURTON, F. W., McKEOWN, G. P. RAYWARD-SMITH, V. J. & SLEEP, M. R. Parallel processing and combinatorial optimization. Presented at the Conference on Combinatorial Optimization, Stirling (1981).

CAMPBELL, H. G., DUDEK, R. A. & SMITH, M. L. A heuristic algorithm for n job, m machine sequencing problem. *Management Sci.,* 16 (1970), 630-637.

CARLIER, J. Probleme a une machine. *Manuscript, Institut de programmation, Universite Paris,* VI (1980).

CHANDRA, R. On n/1/$\bar{F}$ dynamic deterministic systems. *Naval Research Logistics Quarterly,* 26 (1979), 537-544.

CHO, Y. & SAHNI, S. Preemptive scheduling of independent jobs with release and due times on open, flow and job shops. *Technical Report 78-5, Computer Science Department, University of Minnesota, Minneapolis,* (1978).

CHRISTOFIDES, N., MINGOZZI, A. & TOTH, P. State-space relaxation procedures for the computation of bounds to routing problems. *Networks,* 11, (1981), 145-164.

COFFMAN, E. G. (ed.) Computer and job-shop scheduling theory. *Wiley, New York* (1976).

COOK, S. A. The complexity of theorem-proving procedures. *Proc. 3rd Annual ACM Symp. Theory of Computing.* (1971), 151-158.

CONWAY, R. W., MAXWELL, W. L. & MILLER, L. W. Theory of scheduling. *Addison-Wesley, Reading, Massachusetts* (1967).

DANNENBRING, D. G. The evaluation of heuristic solution procedures for large combinatorial problems. *Unpublished Ph.D. dissertation, Columbia University, New York* (1973).

DANNENBRING, D. G. An evaluation of flow-shop sequencing heuristics. *Management Sci.,* 23, No. 11 (1977).

DAY, J. & HOTTENSTEIN, M. P. Review of sequencing research. *Naval Research Logistics Quart.,* 17 (1970), 11-39.

DEKEL, E. & SAHNI, S. Binary Trees and parallel scheduling algorithms. *Department of Computer Science, University of Minnesota, TR* 80-19 (1980).

DEKEL, E. & SAHNI, S. Parallel scheduling algorithms. (Unpublished paper).

DELEEDE, E. & RINNOOY KAN, A. H. G. Unpublished manuscript (1975).

DESSOUKY, M. I. & DEOGUN, J. S. Sequencing jobs with unequal ready times to minimize mean flow-time. *Society for industrial and applied mathematics journal on computing,* 10 (1981), 192-202.

DUDEK, R. A. & TEUTON, O. F. Development of M-stage decision rules for scheduling n jobs through M machines. *Oper. Res.,* 12 (1964), 471-497.

EASTMAN, W. L. A solution to the travelling salesman problem. *Econometrica,* 27 (1959).

EASTMAN, W. L., EVEN, S. & ISAACS, I. M. Bounds for the optimal scheduling of n jobs on m processors. *Management Sci.*, 11, No. 2 (1964).

ELMAGHRABY, S. E. The machine scheduling problem review and extensions. *NRLQ* 15 (1968), 205-232.

EMMONS, H. One machine sequencing to minimize certain functions of jobs tardiness. *Oper. Res.*, 17 (1969), 701-715.

EMMONS, H. One machine sequencing to minimize mean flow time with minimum number tardy. *Naval Res. Logist. Quart.*, 22 (1975), 585-592.

FISHER, M. L. A dual algorithm for the one machine scheduling problem. *Math Programming*, 11 (1976), 229-251.

FISHER, M. L. Lagrangian relaxation methods for combinatorial optimization. *Paper presented at the Summer School in Combinatorial Optimization, Urbino, Italy* (1978), 10-21.

FISHER, M. L. Worst-case analysis of heuristic for scheduling and packing. *Management. Sci.*, 26, No. 1 (1980).

FOX, B. L., LENSTRA, S. K., RINNOOY KAN, A. H. G. & SCHRAGE, L. E. Branching from the largest upper bound: Folklore and facts. *European J. of Oper. Res.*, 2 (1978), 191-194.

GAREY, M. R. Optimal task sequencing with precedence constraints. *Discrete Math.*, 4 (1973), 37-56.

GAREY, M. R. & GRAHAM, R. L. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on computing*, Vol.4 (1975), 187-200.

GAREY, M. R., GRAHAM, R. L. & JOHNSON, D. S. Performance guarantees for scheduling algorithms. *Oper. Res.*, 26, No. 1 (1978).

GAREY, M. R. & JOHNSON, D. S. Scheduling tasks with non-uniform deadlines on two processors. *J.Assoc. Comput. Mach.*, 23 (1976), 461-467.

GAREY, M. R. & JOHNSON, D. S. Computers and intractability: A guide to the theory of NP-completeness. *Freeman, San Francisco* (1979)

GAREY, M. R., JOHNSON, D. S. & SETHI, R. The complexity of flow-shop and job-shop scheduling. *Math. Operations Res.*, 1 (1976), 117-129.

GAREY, M. R., JOHNSON, D. S., SIMONS, B. B. & TARJAN, R. E. Scheduling unit -time tasks with arbitrary release times and deadlines. *J. Comput. System Science*, to appear (1981).

GELDERS, L. & KLEINDORFER, P. R. Co-ordinating aggregate and detailed scheduling decisions in the one-machine job shop: Part I Theory. *Oper. Res.*, 22 (1974), 46-60.

GELDERS, L. & KLEINDORFER, P. R. Co-ordinating aggregate and detailed scheduling in the one machine job shop: Part II Computation and structure. *Oper. Res.*, 23 (1975), 312-324.

GEOFFRION, A. Lagrangian and its uses in integer programming. *Math. Prog. Study*, 2 (1974), 82.

GERE, W. S. Heuristics in job shop scheduling. *Management Science*, 13 (1966), 167-190.

GONZALEZ, T. NP-hard shop problems. Report CS-79-35. *Department of Computer Science, Pennsylvania State University, University Park* (1979).

GONZALEZ, T. & SAHNI, S. Open shop scheduling to minimize finish time. *J.Assoc. Comput. Mach.*, 23 (1976), 665-679.

GONZALEZ, T. & SAHNI, S. Flow shop and job shop schedules: Complexity and approximation. *Oper. Res.*, 26, No. 1 (1978).

GRAHAM, R. L. Bounds for Certain Multiprocessing Anomalies. *Bell Sys. Tech. J.*, 45 (1966), 1563-1581.

GRAHAM, R. L. Bounds on multiprocessing time anomalies. *SIAM J.Appl. Math.*, 17 (1969), 263-269.

GRAHAM, R. L. Bounds on multiprocessing anomalies and related packing anomalies. In: *Proceedings of the 40th AFIPS Conference* (1972).

GRAHAM, R. L., LAWLER, E. L., LENSTRA, J. K. & RINNOOY KAN, A. H. G. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann.Discrete Math.*, 5 (1979), 287-326.

GUPTA, J. N. D. Economic aspects of production scheduling systems. *Journal of the Operations Research Society of Japan*, 13, 169-193.

HARIRI, A. M. A. & POTTS, C. N. An algorithm for single machine sequencing with release dates to minimize total weighted completion time. *Report BW 143, Mathematisch Centrum, Amsterdam* (1981).

HELD, M. & KARP, R. M. A dynamic programming approach to sequencing problems. *J. SIAM*, 10 (1962).

HELLER, J. Combinatorial, probabilistic and statistical aspects of an MxJ scheduling problem. *AEC Research and Development Report, New York* (1959).

HELLER, J. Some numerical experiments for an MxJ flow-shop and its decision theoretical aspects. *Oper. Res.*, 8 (1960), 178-184.

HORN, W. A. Single-machine job sequencing with tree like precedence ordering and linear delay penalties. *SIAM, Journal on Applied Mathematics*, 23 (1972), 189-202.

IGNALL, E. & SCHRAGE, L. Application of the branch and bound technique to some flow-shop scheduling problems. *Oper. Res.*, 13 (1965), 400-412.

JACKSON, J. R. Scheduling a production line to minimize maximum tardiness. *Research Report 43, Management Science Research Project, University of California, Los Angeles* (1955).

JACKSON, J. R. An extension of Johnson's results on job lot scheduling. *Naval Res. Logist. Quart.*, 3 (1956), 201-203.

JOHNSON, S. M. Optimal two- and three-stage production schedules with setup times included. *Naval Res. Log. Quart.*, 1 (1954), 61-68.

KARP, R. M. Reducibility among combinatorial problems. In: *R. E. Miller & J. W. Thatcher (eds.) Complexity of computer computations, Plenum Press, New York* (1972), 85-103.

KISE, H., IBARAKI, T. & MINE, H. Approximate algorithms for the one-machine maximum lateness scheduling problem with ready times. *Technical Report, Department of Applied Mathematics and Physics, Kyoto University, Kyoto, Japan* (1978A).

KISE, H., IBARAKI, T. & MINE, H. A solvable case of the one-machine scheduling problem with ready and due times. *Oper. Res.*, 26 (1978B), 121-126.

KISE, H. & UNO, M. One-machine scheduling problems with earliest start and due time constraints. *Mem. Kyoto Tech. Univ. Sci. Tech.*, 27 (1978), 25-34.

KURISU, T. Two-machine scheduling under required precedence among jobs. *Operations Research Soc. of Japan*, 20 (1976).

KURISU, T. Two-machine scheduling under arbitrary precedence constraints. *Oper. Res. Society of Japan*, 20, No. 2 (1977).

LABETOULLE, J., LAWLER, E. L., LENSTRA, J. K. & RINNOOY KAN, A. H. G. Preemptive scheduling of uniform processors subject to release dates. *Report BW 99, Mathematisch Centrum, Amsterdam* (1979).

LAGEWEG, B. J., LENSTRA, J. K. & RINNOOY KAN, A. H. G. Minimizing maximum lateness on one machine: Computational Experience and Some Applications. *Statist. Neerlandica*, 30 (1976), 25-41.

LAGEWEG, B. J., LENSTRA, J. K. & RINNOOY KAN, A. H. G. Job-shop scheduling by implicit enumeration. *Management Sci.*, 24 (1977), 441-450.

LAGEWEG, B. J., LENSTRA, J. K. & RINNOOY KAN, A. H. G. A general bounding scheme for the permutation flow-shop problem. *Oper. Res.*, 26, No. 1, (1978).

LAND, A. H. & DOIG, A. G. An automatic method for solving discrete programming problems. *Econometrica*, 28 (1960), 497-520.

LAWLER, E. L. Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19 (1973), 544-546.

LAWLER, E. L. Sequencing to minimize the weighted number of tardy jobs. *Rairo Rech. Oper., 10.5 Suppl.* (1976), 27-33.

LAWLER, E. L. A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. *Ann. Discrete Math.*, 1 (1977), 331-342.

LAWLER, E. L. Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics*, 2 (1978), 75-90.

LAWLER, E. L. Efficient implementation of dynamic programming algorithms. To appear (1981).

LAWLER, E. L. Unpublished work (1981A).

LAWLER, E. L., LENSTRA, J. K. & RINNOOY KAN, A. H. G. Minimizing maximum lateness in a two-machine open shop. *Oper. Res.*, 6 (1981A), 153-158.

LAWLER, E. L., LENSTRA, J. K. & RINNOOY KAN, A. H. G. Recent developments in deterministic sequencing and scheduling: A survey. To appear (1981).

LAWLER, E. L. & MOORE, J. M. A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16 (1969), 77-84.

LAWLER, E. L., TARJAN, R. E. & VALDES, J. Analysis and isomorphism of series parallel digraph. (To appear).

LENSTRA, J. K. Sequencing by enumerative methods. *Mathematical Centre Tracts 69, Mathematisch Centrum, Amsterdam* (1977).

LENSTRA, J. K. Unpublished work (1981).

LENSTRA, J. K. & RINNOOY KAN, A. H. G. A recursive approach to the generation of combinatorial configurations. *Working Paper WP/75/25, Graduate School of Management, Delft* (1975).

LENSTRA, J. K. & RINNOOY KAN, A. H. G. Complexity of scheduling under precedence constraints. *Oper. Res.*, 26 (1978), 22-35.

LENSTRA, J. K. & RINNOOY KAN, A. H. G. Computational complexity of discrete optimization problems. *Ann. Discrete Math.*, 4 (1979), 121-140.

LENSTRA, J. K. & RINNOOY KAN, A. H. G. Complexity results for scheduling chains on a single machine. *European J. Oper. Res.*, 4 (1980), 270-275.

LENSTRA, J. K., RINNOOY KAN, A. H. G. & BRUCKER, P. Complexity of machine scheduling problems. *Ann. Discrete Math.*, 1 (1977), 343-362.

LIU, C. L. Optimal scheduling on multiprocessor computing systems. In: *Proceedings of the 13th Annual Symposium on Switching and Automata Theory* (1972).

LOMNICKI, Z.A. A branch and bound algorithm for the exact solution of the three machine scheduling problem. *Oper. Res.*, 16, No. 1 (1965), 89-100.

McMAHON, G. B. Optimal production schedules for flow shops. *Journal of the Canadian Oper. Res. Society*, 7 (1969), 141-151.

McMAHON, G. B. A study of algorithms for industrial scheduling problems. *Ph.D. Thesis, University of New South Wales* (1971).

McMAHON, G. B. & BURTON, P. G. Flow-shop scheduling with the branch and bound method. *Oper. Res.*, 15 (1967), 473-481.

McMAHON, G. B. & FLORIAN, M. On scheduling with ready times and due dates to minimize maximum lateness. *Oper. Res.*, 23 (1975), 475-482.

MITTEN, L. G. Sequencing jobs on two machines with arbitrary time lags. *Management Sci.*, 5 (1959A).

MITTEN, L. G. A scheduling problem. *J.Ind.Eng.*, 10 (1959B).

MONMA, C. L. The two-machine maximum flow time problem with series parallel precedence constraints: An algorithm and extensions. *Oper. Res.*, 27, No. 4 (1979).

MONMA, C. L. Sequencing to minimize the maximum job cost. *Oper. Res.*, 28, No. 4 (1980).

MONMA, C. L. Sequencing with general precedence constraints. (To appear).

MONMA, C. L. & SIDNEY, J. B. Sequencing with series parallel precedence constraints. *Mathematics of Operations Res.*, 4, No. 3 (1979).

MOORE, J. M. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15 (1968), 102-109.

MORTON, T. E. & DHARAN, B. G. Algoristics for single machine sequencing with precedence constraints, *Management Science*, 24, No. 10 (1978).

MULLER-MERBACH, H. Heuristics and their design: A survey. *European J. of Oper.Res.*, 8 (1981), 1-23.

MUTH, J. F. & THOMPSON, G. L. (eds.) Industrial scheduling. *Prentice-Hall, Englewood, N.J.* (1963).

NABESHIMA, I. On the bound of makespans and its application in M machine scheduling problem. *J. Operations Res. Soc. of Japan*, 9 (1967), 98-136.

PALMER, D. S. Sequencing jobs through a multi-stage process in minimum total time: A quick method of obtaining a near optimum. *Oper. Res. Quart.*, 16, No. 1 (1965).

PICARD, J. C. & QUEYRANNE, M. The time-dependent travelling salesman problem and its application to the tardiness problem in one-machine scheduling. *Oper. Res.*, 26 (1978), 86-110.

POTTS, C. N. The job-machine scheduling problem, *Ph.D. Thesis, University of Birmingham.* (1974)

POTTS, C. N. An adaptive branching rule for the permutation flow-shop problem. *European J. Oper. Res.*, 5 (1980A), 19-25.

POTTS, C. N. Analysis of a heuristic for one machine sequencing with release dates and delivery times. *Oper. Res.*, 28 (1980B), 1436-1441.

POTTS, C. N. An algorithm for the single machine sequencing problem with precedence constraints. In: *Mathematical Programming Study 13 on Combinatorial Optimization*, II (1980C), 78-87.

POTTS, C. N. Private communication (1981).

RINALDI, G. & SASSANO, A. On a job scheduling problem with different ready times: Some properties and a new algorithm to determine the optimal solution. *Report R 7724, Instituto di Automatica, University di Rome* (1977).

RINNOOY KAN, A. H. G. Machine scheduling problems: Classification, complexity and computations. *Martinus Nyhoff, The Hague,* (1976).

RINNOOY KAN, A. H. G., LAGEWEG, B. J. & LENSTRA, J. K. Minimizing total costs in one-machine scheduling. *Oper. Res.,* 23, No. 5 (1975).

SAHNI, S. Algorithms for scheduling independent tasks. *J.Assoc. Comput. Mach.,* 23 (1976), 116-127.

SCHRAGE, L. Obtaining optimal solutions to resource constrained network scheduling problems. Unpublished manuscript (1971).

SIDNEY, J. B. An extension of Moore's due date algorithm. In: *S. E. Elmaghraby (ed.), Symposium on the theory of scheduling and its application, Lecture Notes in Economics and Mathematical Systems 86, Springer, Berlin* (1973), 393-398.

SIDNEY, J. B. Decomposition algorithms for single-machine sequencing with precedence relations and deferal costs. *Oper. Res.,* 22 (1975), 283-298.

SIDNEY, J. B. The two-machine maximum flow time problem with series-parallel precedence relations. *Oper. Res.,* 27, No. 4 (1979).

SIMONS, B. A fast algorithm for single processor scheduling. *Proc. 19th Annual IEEE Symp. Foundations of Computer Science* (1978), 246-252.

SMITH, W. E. Various optimizers for single-stage production. *Naval Res. Logist. Quart.,* 3 (1956), 59-66.

SMITH, R. D. & DUDEK, R. A. A general algorithm for solution of the n-job M-machine sequencing problem of the flow-shop. *Oper. Res.,* 15 (1967), 71-82.

SMITH, R. D. & DUDEK, R. A. Erratum. *Oper. Res.,* 17 (1969).

SRINIVASON, V. A hybrid algorithm for the one machine sequencing problem to minimize total tardiness. *Naval Res. Logist. Quart.,* 18 (1971), 317-327.

SZWARC, W. Elimination methods in the mxn sequencing problem. *Naval Research Logistics Quarter,* 18 (1971), 295-305.

SZWARC, W. Optimal elimination methods in the mxn flow-shop scheduling problem. *Oper. Res.,* 21 (1973), 1250-1259.

SZWARC, W. Remarks. *Oper. Res.,* 23 (1975).

TOWNSEND, W. The single machine problem with quadratic penalty function of completion times: A branch-and-bound solution. *Management Science,* 24 (1978), 530-534.

VAN WASSENHOVE, L. Special-purpose algorithms for one machine sequencing problems with single and composite objectives. *Ph.D. Thesis, Industrial Beleid, Katholicke Universiteit Leuven* (1979).

VAN WASSENHOVE, L. & GELDERS, L. F. Solving a bicriterion scheduling problem. *European J. of Oper. Res.,* 4 (1980), 42-48.

WILKERSON, L. J. & IRWIN, J. D. An improved algorithm for scheduling independent tasks. *AIIE Transactions,* 3, No. 3 (1971).