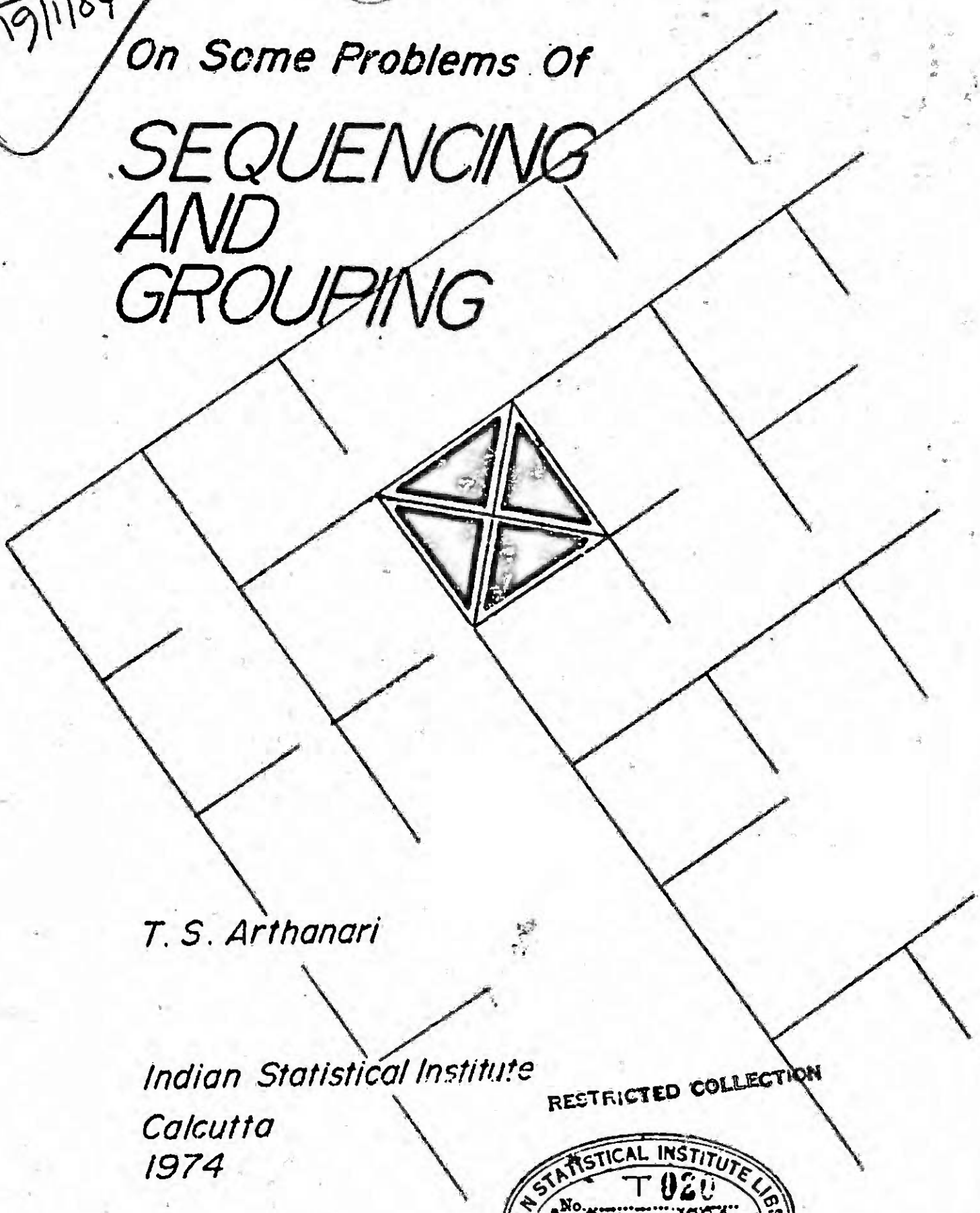
On Some Problems Of

SEQUENCING AND GROUPING



T. S. Arthanari

Indian Statistical Institute
Calcutta
1974

On Some Problems Of SEQUENCING AND GROUPING

T. S. Arthanari

A thesis submitted to
the Indian Statistical Institute
in partial fulfilment of
the requirements for
the degree of Doctor of Philosophy

Calcutta
1974

ACKNOWLEDGEMENT

I thank the SQC Division of the Indian Statistical Institute, for all the facilities extended to me to carry out the research work. I thank Professor T.V.Hanurav, Dean of Studies and Professor J. Roy, Head, Computing Science Unit for allowing me to use the facilities available in their offices.

I am particularly indebted to Dr. A.R.Rao, my thesis supervisor, who went through the thesis meticulously and offered many valuable suggestions. His willingness to read promptly many drafts of this thesis greatly enhanced my progress.

I express my boundless thanks to Professor K.G. Ramamurthy, for encouraging me to work in this branch of Operations Research and permitting me to include the materials from some of our joint papers.

I thank, my colleague and friend Mr. S.R.Mohan for the stimulating discussions and for allowing me to include our joint work.

I thank, Mr. C.H. Sastry for the programming help rendered by him.

I am greatful to all my colleagues and friends at SQC Division for their cooperation and constant encouragement throughout the work. Especially, I owe my thanks to Messrs. A.C.Mukhopadhyay, R.K.Chordia, R.J.Pandey, V.V.Buche, A. Majumdar, J. Sharma, S. Guha and Chitta Babu.

At each stage of preparation of this thesis my friend Acchu (Mr. N.R.Achuthan) rendered such help with affection and interest that I cannot possibly express my indebtedness to him in a few words. He is a gem of a friend.

I thank Mr. Dhruba Roy for designing an impressive cover for the thesis.

I appreciate and thank Mr. Arun Das for the quick, excellent and intelligent typing of this thesis.

CONTENTS

Contents

ooooOoooo

Some people shave before bathing,

And about people who bathe before shaving they are scathing,

While those who bathe before shaving,

Well, they imply that those who shave before bathing are misbehaving.

Suppose you shave before bathing, well the advantage is that you don't have to make a special job of washing the lather off afterwards, it just floats off with the rest of your accumulations in the tub,

But the disadvantage is that before bathing your skin is hard and dry and your beard confronts the razor like a grizzly bear defending its cub.

Well then, suppose you bathe before shaving, well the advantage is that after bathing your skin is soft and moist, and your beard positively begs for the blade,

But the disadvantage is that to get the lather off you have to wash your face all over again at the basin almost immediately after washing it in the tub, which is a duplication of effort that leaves me spotless but dismayed.

The referee reports, gentlemen, that Fate has loaded the dice,

Since your only choice is between walking around all day with a sore chin or washing your face twice,

So I will now go and get a shave from a smug man in a crisp white coat,

And I will disrupt his smugness by asking him about his private life, does he bathe before shaving or shave before bathing, and then I will die either of laughing or of a clean cut throat.

- Ogden Nash

I N T R O D U C T I O N

The problems of sequencing arise in almost all walks of life. Theory of scheduling deals with such problems. Usually, these problems are stated in the literature in terms of jobs, machines, operations, penalties, due dates et cetera, that is, in the language of machine - shops. The real life problems of machine - shop job sequencing are of a complex nature. In general, we consider processing n items on a certain group of machines, so as to optimize certain objective, subject to various constraints on precedence, machine availability, due date and so on. The job sequencing problems are included in a general class of problems known as resource constrained network problems. These are the well known CPM/PERT type network problems with constraints on available resources.

Johnson's (1954) paper initiated work in developing and analysing mathematical models representing machine - shops which was continued by Wagner (1959), Bowman (1959), Manne (1960), Dantzig (1960), Held and Karp (1962), Brown and Lomnicki (1966), to name a few. In the past nineteen years many researchers have contributed to the growth of the sequencing theory. A review of the work done upto 1968 is available in Elmaghraby (1968). Another review has been done by Day and Hottenstein (1970). A survey of the methods proposed for the sequencing

problems with the objective of minimising total elapsed time is given in Bakshi and Arora (1969).

In the recent past several papers have appeared on the general machine - shop problems. Balas (1969) has given a disjunctive graph approach to sequencing problems, in which several critical path subproblems are solved to find an optimal solution to the problem using an implicit enumeration method. Death and Charlton (1970), Nabashima (1971), Florian and others (1971), Schrage (1970) have also given procedures for solving general scheduling problems. Fisher (1973) gives a Lagrangian approach, to the resource-constrained scheduling problem, in which he uses Lagrangian multipliers to find bounds and uses the bounds in a Branch and Bound algorithm.

Even though the general problem has many algorithms, it is of interest to consider some particular cases with certain assumptions, as efficient algorithms are then possible exploiting the simplicity arising out of these assumptions. The Flow-shop problems under some usual assumptions have been considered by many authors with various objectives (see Elmaghraby (1968), Szwarc (1971), Maxwell and others (1967)). Inspite of the fact the problems studied are very much restrictive under such assumptions, the study of such problems is no less important because the method of attack for simplified systems may be

useful in solving more complex ones and these simplified problems are themselves interesting research problems.

In this work we consider mainly the flow-shop situation under the following assumptions.

A.1 A set N of n jobs must be processed.

A.2 All the machines and jobs are available for processing at time 0.

A.3 At any given instant of time, on any machine, processing can go on for one and only one job. Also a job cannot be processed by more than one machine at any point of time.

A.4 No preemption of processing of any job on any machine is allowed, that is, an operation once started must be completed without interruption.

A.5 The machines are available continuously for processing (that is, no break down of machines is considered) until all the jobs are completed.

A.6 A known finite time is required to process a job on a machine. These processing times are independent of the order of processing.

A.7 There is only one machine at each 'stage of processing'.

The order in which a job must go through various stages is generally known as the technological ordering. If all the jobs have the same technological ordering the term flow-shop is used to describe such a situation. We assume there are m stages and the jobs are processed first at Stage 1 and then at Stage 2 and so on.

In addition we make the assumption given below.

A.8 No passing is allowed. This means that if job j is completed before job k at Stage 1 then job j is completed before job k in all the stages. And so, we need to consider only one ordering among the jobs for all the stages.

We use Maxwell and others' (1967) four parameter notation to identify a problem under consideration as follows :

((1) / (2) / (3) / (4))

where,

(1) -- In the first place we give the number of jobs under consideration.

(2) -- Secondly we give, the number of stages of processing. If we relax the assumption A.7 we instead give a vector $(k_1, k_2, \ldots, k_m)$ giving the number of machines at each stage. For example, m denotes that there are m stages with one machine

at each stage while (m) denotes that there is only one stage with m machines in that stage.

(3) -- In the third place, we describe the type of shop under consideration, that is, the technological routing of the jobs through the machines. We write F for a flow-shop. For a randomly routed shop we use R and for job-shop we use J and so on.

(4) -- Finally, the objective function that is minimised is given in the fourth place. We use F_{max} for total elapsed time, $\bar{C}$ for average completion time, T_u for weighted tardiness, u(T) for any general real valued function of tardiness, C_u for weighted sum of completion times, $\bar{T}$ for average tardiness and f for any general objective.

For example,

(i) $(n/(m,1)/F/F_{max})$ denotes a problem in which n jobs are to be processed through 2 stages, first in Stage 1 and then in Stage 2, so as to minimize the total elapsed time, when there are m machines in Stage 1 under the usual assumptions excepting A.7.

(ii) $(n/2/F/F_{max})$ refers to Johnson's two stage problem.

In general we employ the notations used in the book 'Theory of Scheduling' by Maxwell and others (1967). A list of symbols commonly used is given at the end of the thesis.

We next give an outline of the results of this thesis chapterwise.

In chapter 1 we consider some special cases of the $(n/m/F/F_{max})$ problem and introduce certain cumulative dominance conditions. The theorems proved under these conditions are useful in reducing the problems to smaller sized problems. Next we consider $(n/(m,1)/F/F_{max})$ problem and give a Branch and Bound algorithm. Branching effort is reduced by showing that there exists an optimal schedule belonging to a subset of all possible schedules, called the set of preferred schedules. A near-optimal rule is given. In section 1.3 we extend Elmaghraby's (1971) convex graph definition to general switching rules. Also we show that Rau's (1971) conditions A and B are equivalent to some simpler conditions and B imples A. Chapter 1 ends with a 'good' algorithm for finding an optimal sequence when CBS-rule is applicable.

Chapter 2 deals with the $(n/m/F/f)$ problem. A general dominance theorem is proved for any f having a certain property. The Branch and Bound algorithm for the $(n/m/F/F_{max})$ problem given by Brown and Lomnicki (1966) is modified for the $(n/m/F/u(T))$ problem with nondecreasing $u(T)$. For some special objective functions $u(T)$, using the solution to the assignment problem

with the cost matrix having monotone property, certain simple lower bounds are obtained for $u(T)$ for any completion of a partial sequence. Next we give simple rules to solve two special cases of the $(n/2/F/C_u)$ problem. In section 2.4, we extend the theorems proved for the $(n/m/F/F_{max})$ problem with the cumulative dominance conditions in section 1.1 to the $(n/m/F/f)$ problem with f any regular measure of performance. The $(n/1/F/T_u)$ problem is considered in section 2.5 and the results of Hamilton Emmons (1969) are extended to this case. The same problem with common due date is discussed in section 2.6. In section 2.7, the $(n/1/F/f)$ problem to minimise the number of late jobs is considered when the jobs have a common due date and different arrival times. It is shown that the problem can be solved by solving an equivalent $(n/1/F/f)$ problem to minimise the number of late jobs with different due dates, the jobs being available simultaneously.

In chapter 3, we consider some grouping problems, two of them scheduling problems and the third the cluster analysis problem. We consider the $(n/(m)/F/C_u)$ problem with nonidentical machines and give a partial enumeration algorithm. A few bounds are suggested. The algorithm is simplified for the identical machines case. In section 3.2, we consider the problem of

sequecing n jobs through m different machines to minimise the total elapsed time, in the absence of technological constraints and with the restriction that all the jobs have to be processed through all the machines once and only once. When the jobs are identical, the problem is formulated as a problem of grouping the items into m groups and finding an order of processing for each group. When n is sufficiently large optimal solutions are obtained without resorting to the general algorithm which requires partial enumeration. In section 3.3, we apply the algorithm developed in section 3.1 to the cluster analysis problem.

C H A P T E R 1

FLOW SHOP PROBLEMS-I (MINIMIZATION OF TOTAL ELAPSED TIME)

1.0 Introduction :

A flow-shop is one in which all the jobs follow essentially the same path from one stage to another. Johnson's paper (1954) in which he considers all jobs having the same technological ordering A,B through the stages A and B, initiated work in this direction. Johnson (1954) and Bellman (1956) have used 'contiguous binary switching' and dynamic programming respectively for solving $(n/2/F/F_{max})$ problem. Johnson also solves certain special cases of the $(n/3/F/F_{max})$ problem. Wagner and Story (1963) have considered an integer programming formulation of the $(n/3/F/F_{max})$ problem. Lomnicki (1965) has given a Branch and Bound method for solving the three stage flow-shop problem. Ignall and Schrage (1965) have also applied Branch and Bound method to this problem. Also they consider the problem of minimising mean flow time in a two stage flow-shop, i.e., $(n/2/F/\bar{F})$ problem. Jackson (1956) solves $(n/(1,n,1)/F/F_{max})$ problem and shows that the optimal sequence obtained by Johnson in three stage special cases turns out to be optimal in this problem also. Szwarc (1968) considers several special cases of $(n/3/F/F_{max})$ problem. Arthanari and Mukhopadyay (1971) have solved two special cases of $(n/3/F/F_{max})$ problem.

Lomnicki and Brown (1966) have given a Branch and Bound method extending the ideas from Lomnicki (1965) for solving $(n/m/F/F_{max})$ problem, with the no passing restriction. Mc Mahon and Burton (1967) also give a partial enumeration method for the same problem. Nabashima(1967) has given improved bounds for the $(n/m/F/F_{max})$ problem making use of Johnson's result for the two stage problem. Balas (1969) has introduced a disjunctive graph approach for the sequencing problem. Charlton and Death (1970), Nabashima (1971) give partial enumeration algorithms considering the disjunctive graph formulation.

A summary of the empirical analysis of certain flow-shop problems by Heller (1960) and Nugent (1964) is given in the book Maxwell and others(1967). Wagner and Giglio (1964), Gupta (1971a), Ashour (1970) have also used such approaches to flow-shop problems. Gupta uses experimental analysis for studying different criteria used as objectives. Ashour has done a statistical study to compare complete enumeration against decomposition procedures, in solving flow-shop problems.

Dudek and Teuton (1964) have initiated the work in finding 'elimination criteria' for excluding certain sequences from the search for optimal sequence. Smith and Dudek (1967), Bagga and Chakravarti (1968), Gupta (1971'), Ignall and Schrage (1965) and Szwarc (1971) have given different elimination criteria. Szwarc (1973) shows the non-optimality of some of these criteria, and

shows his criterion to be stronger than that given by Smith and Dudek and equivalent to that given by Gupta.

Elmaghraby (1971) gives a graph theoretic interpretation of the theorem by Smith (1956) which gives a sufficient condition for contiguous binary switching to be optimal. Rau (1971) solves the problem of minimizing a certain function over permutations of n integers, using contiguous binary switching.

With this introduction to the work that has been done in this area, we next explain what is done in this chapter. Section 1 of this chapter is devoted to results pertaining to several special cases of the m-stage problem, namely $(n/m/F/F_{max})$ problem. Under certain dominance conditions, two useful theorems are proved. Section 2 discusses a simple hybrid case of the two stage Johnson's problem, Ramamurthy and Arthanari (1971). A near-optimal-rule is suggested, and 174 problems of different sizes are solved using the near-optimal-rule. It is found that in over 94 per cent of the cases the total elapsed time found was within 5 per cent of the optimum. Incorrectness of Bagga and Mittal's (1973) assertions is shown considering two counter examples given by Ramamurthy (1973). Section 3 deals with some miscellaneous observations made regarding CBS-rules, John G. Rau's paper (1971) and the switching rules in general.

1.1 Some Special Cases of the $(n/m/F/F_{max})$ problem

In this section some special cases of the m-stage flow-shop problem are considered, with the 'no passing' restriction. The objective is to minimize the total elapsed time. The process order for each job is the same and is $M_1, M_2, \ldots, M_m$. Let p_{ij} = processing time for job 'j' on machine M_i, $1 \leq i \leq m$; $1 \leq j \leq n$. Let $\bar{p} = (j_1 \ldots j_n)$ and let $g(\bar{p})$ be the idle time on the last machine, for $\bar{p}$. Cases I and II discussed below are known for $m = 3$, Johnson(1954). Case V is known for $m = 3$, Szwarc (1968). Cases III and IV for $m = 3$ are considered by Arthanari and Mukhopadyay (1971).

Case I $\quad \min_j p_{1j} \geq \max_j \sum_{r=2}^{m-1} p_{rj}$

Let X_i denote the idle time on the last machine immediately before job j_i while using the sequence $\bar{p}$. Let $t((j_1 \ldots j_i), k)$ denote the time required to complete the jobs $(j_1 \ldots j_i)$ in that order on the machines $M_1, \ldots, M_k$. Then

$$X_1 = \sum_{r=1}^{m-1} p_{rj_1}$$

$$X_2 = \max[t((j_1\ j_2), m-1) - p_{mj_1} - X_1,\ 0]$$

But $t((j_1 j_2 \ldots j_i), m-1) = t((j_1 \ldots j_{i-1}), 1) + \sum_{r=1}^{m-1} p_{rj_i}$

(since $\min_j p_{1j} \geq \max_j \sum_{r=2}^{m-1} p_{rj}$)

$$= \sum_{s=1}^{i-1} p_{1j_s} + \sum_{r=1}^{m-1} p_{rj_i}$$

So

$$X_2 = \max [p_{1j_s} + \sum_{r=1}^{m-1} p_{rj_2} - p_{mj_1} - X_1, 0]$$

Similarly

$$X_i = \max[\sum_{s=1}^{i-1} p_{1j_s} + \sum_{r=1}^{m-1} p_{rj_i} - \sum_{s=1}^{i-1} p_{mj_s} - \sum_{s=1}^{i-1} X_s , 0]$$

Therefore

$$\sum_{s=1}^{i} X_s = \max[\sum_{s=1}^{i-1} p_{1j_s} + \sum_{r=1}^{m-1} p_{rj_i} - \sum_{s=1}^{i-1} p_{mj_s}, \sum_{s=1}^{i-1} X_s]$$

Adding and subtracting $\sum_{s=1}^{i-1} \sum_{r=2}^{m-1} p_{rj_s}$ to the first term inside the square brackets in the above expression for $\sum_{s=1}^{i} X_s$, we have,

$$\sum_{s=1}^{i} X_s = \max \{ \sum_{s=1}^{i} [\sum_{r-1}^{m-1} p_{rj_s}] - \sum_{s=1}^{i-1} [\sum_{r=2}^{m} p_{rj_s}], \sum_{s=1}^{i-1} X_s \}$$

Using this expression for different values of i, we get $g(\bar{p})$, the idle time on the last machine as

$$g(\bar{p}) = \sum_{u=1}^{n} X_u = \max_{1 \leq u \leq n} \{ \sum_{s=1}^{u} [\sum_{r=1}^{m-1} p_{rj_s}] - \sum_{s=1}^{u-1} [\sum_{r=2}^{m} p_{rj_s}] \}$$

Thus to get an optimal solution to the $(n/m/F/F_{max})$ problem when case I holds, Johnson's two stage rule can be applied to the problem $(n/2/F/F_{max})$ with $A_j = \sum_{r=1}^{m-1} p_{rj}$ and $B_j = \sum_{r=2}^{m} p_{rj}$, $1 \leq j \leq n$.

Example 1.11: The following data gives the p_{kj}'s for a $(4/5/F/F_{max})$ problem.

jobs j \ Machines k	1	2	3	4	5
1	6	1	2	1	7
2	7	2	1	1	4
3	5	1	1	2	6
4	4	1	1	2	1

We find $\min_j p_{1j} = 4 \geq \max_j (\sum_{r=2}^{4} p_{rj}) = 4$. So case I holds.

The corresponding two-stage problem is

j	A_j	B_j
1	10	11
2	11	8
3	9	10
4	8	5

Using Johnson's rule $\bar{p} = (3,1,2,4)$ is found to be optimal and $g(\bar{p}) = 9$. Thus the optimal total elapsed time is $9 + 18 = 27$, since 18 is the total time required to process all the jobs on the 5th machine.

Case II. $\quad \min_j p_{mj} \geq \max_j \left[\sum_{r=2}^{m-1} p_{rj} \right]$

In this case, we find X_i in a slightly different manner. Consider $\sum_{s=1}^{i} p_{1j_s}$. There are two possibilities.

Case a : $\quad t((j_1 \ldots j_{i-1}), m-1) \geq \sum_{s=1}^{i} p_{1j_s}$. Let $S_{m,i-1}$ be the starting time of job j_{i-1} on machine m. Now, it is easy to see

$$t((j_1 \ldots j_i), m-1) \leq t((j_1 \ldots j_{i-1}), m-1) + \sum_{r=2}^{m-1} p_{rj_i}$$

$$\leq t((j_1 \ldots j_{i-1}), m-1) + p_{mj_{i-1}}$$

by Case II

$$\leq S_{m,i-1} + p_{mj_{i-1}}$$

$$= t((j_1 \ldots j_{i-1}), m).$$

Hence, under case a,

$$X_i = 0.$$

Case b : $\quad t((j_1 \ldots j_{i-1}), m-1) < \sum_{s=1}^{i} p_{1j_s}$

Then

$$t((j_1 \ldots j_i), m-1) = \sum_{s=1}^{i} p_{1j_s} + \sum_{r=2}^{m-1} p_{rj_i}$$

Hence under Case b,

$$X_i = \text{Max}\Big[\sum_{s=1}^{i-1} p_{1j_s} + \sum_{r=1}^{m-1} p_{rj_i} - \sum_{s=1}^{i-1} X_s - \sum_{s=1}^{i-1} p_{mj_s},\ 0\Big]$$

Thus in both the cases a and b we have

$$X_i = \text{Max}\Big[\sum_{s=1}^{i-1} p_{1j_s} + \sum_{r=1}^{m-1} p_{rj_i} - \sum_{s=1}^{i-1} p_{mj_s} - \sum_{s=1}^{i-1} X_s,\ 0\Big]$$

Now it is immediate that the $g(\bar{p})$ is same as in Case I. So the same procedure is applicable in this case also.

Case III : $\quad \underset{j}{\text{Max}}\Big[\sum_{r=1}^{m-2} p_{rj}\Big] \leq \underset{j}{\text{Min}}\ p_{m-1\ j}$

We have then

$$g(\bar{p}) = \underset{1\leq u\leq n}{\text{Max}}\ \Big[\sum_{r=1}^{m-2} p_{rj_1} + \sum_{s=1}^{u} p_{m-1\ j_s} - \sum_{s=1}^{u-1} p_{mj_s}\Big]$$

Let $\quad \omega(j) = \sum_{r=1}^{m-2} p_{rj} + p_{m-1\ j} - p_{mj}$

then,

$$g(\bar{p}) = \omega(j_1) + \text{Max}\Big[p_{mj_1},\ \underset{2\leq u\leq n}{\text{Max}}\ \Big(\sum_{s=2}^{u} p_{m-1\ j_s} - \sum_{s=2}^{u-1} p_{mj_s}\Big)\Big]$$

Let $\underline{P}_j$ denote the set of all sequences with $j_1 = j$.

Let $\quad I_j = \underset{\bar{p}\ \varepsilon\ \underline{P}_j}{\text{Min}}\ \Big[\underset{2\leq u\leq n}{\text{Max}}\ \Big(\sum_{s=2}^{u} p_{m-1\ j_s} - \sum_{s=2}^{u-1} p_{mj_s}\Big)\Big]$

I_j can be easily found using Johnson's sequence of the (n-1) jobs other than j on the last two machines, M_{m-1} and M_m.

Hence,

$$\underset{\bar{p}}{\text{Min}}\ g(\bar{p}) = \underset{1\le j\le n}{\text{Min}}\ [\omega(j) + \text{Max}(I_j,\ p_{mj})] \qquad \ldots 1.1.1$$

Let D_j be the expression within square brackets in 1.1.1.

Let S_j be an optimal sequence corresponding to I_j. Then $(j_o,\ S_{j_o})$ is optimal for the $(n/m/F/F_{max})$ problem, where j_o is such that $D_{j_o} = \underset{1\le j\le n}{\text{Min}}\ D_j$.

Example 1.1.2: The following data gives the p_{kj}'s for a $(4/6/F/F_{max})$ problem.

j \ k	1	2	3	4	5	6
1	1	1	5	2	10	8
2	4	2	1	1	9	10
3	2	2	2	1	14	6
4	2	3	2	2	9	12

We find, $\underset{j}{\text{Min}}\ p_{m-1\ j} = 9 \ge \underset{j}{\text{Max}} \sum_{r=1}^{m-2} p_{rj} = 9$, so Case III holds.

Using Johnson's rule,

$I_1 = 10$ and $S_1 = (423)$

$I_2 = 13$ and $S_2 = (413)$

$I_3 = 9$ and $S_3 = (421)$

$I_4 = 15$ and $S_4 = (213)$

I_i can be found using the expression for idle time on the last machine in the two stage problem. Now

$$\omega(j) = \sum_{r=1}^{m-1} p_{rj} - p_{mj} .$$

So

$$\omega(1) = 11$$

$$\omega(2) = 7$$

$$\omega(3) = 15$$

and $$\omega(4) = 6$$

Next $D_j = \omega(j) + \text{Max}(I_j, p_{mj})$ is found for every j. We have

$$D_1 = 21, \quad D_2 = 20, \quad D_3 = 24 \quad \text{and} \quad D_4 = 21$$

Since D_2 is the minimum over D_j's, (2413) is an optimal sequence.

Case IV : $$\text{Max}_j \left(\sum_{r=3}^{m} p_{rj} \right) \leq \text{Min}_j \, p_{2j}$$

In this case we fix job j as the last job and solve a corresponding two machine problem with first two machines and the (n-1) jobs other than j. Let S_j be a Johnson's sequence corresponding to that problem with I_j as the idle time on the second machine. Then (S_{j_o}, j_o) is an optimal sequence where j_o is such that $D_{j_o} = \text{Min}_j D_j$, $D_j = \text{Max}(I_j, a-b+p_{2j}] + \sum_{r=3}^{m} p_{rj}$ with $a = \sum_{j=1}^{n} p_{1j}$ and $b = \sum_{j=1}^{n} p_{2j}$. The analysis is similar to that of Case III with slight modifications.

Case V : $\max_j \left(\sum_{r=k+1}^{m} p_{rj}, \sum_{r=1}^{k-1} p_{rj} \right) \leq \min_j p_{kj}$

for some k, $2 \leq k \leq m-1$

Then we have the total elapsed time

$$F_{max}(\bar{p}) = \sum_{r=1}^{k-1} p_{rj_1} + \sum_{s=1}^{n} p_{kj_s} + \sum_{r=k+1}^{m} p_{rj_n}$$

Therefore

$$\min_{\bar{p}} F_{max}(\bar{p}) = \min_{\substack{1 \leq s, q \leq n \\ s \neq q}} \left[\sum_{r=1}^{k-1} p_{rs} + \sum_{r=k+1}^{m} p_{rq} \right] + \sum_{j=1}^{n} p_{kj} \quad \ldots\ldots 1.1.2$$

Let s_o, q_o be such that the minimum on the R.H.S. in 1.1.2 is attained for s_o and q_o. Then, $(s_o, j_2 \ldots j_{n-1}, q_o)$ is optimal, where $(j_2 \ldots j_{n-1})$ is any permutation of the $(n-2)$ jobs other than s_o and q_o.

Now we identify several other cases in which the $(n/m/F/F_{max})$ problem can be reduced to that of solving one or more smaller sized problems.

Definition 1.1.1 Backward Cumulative Dominance(BCD). We say BCD conditions are satisfied for machine M_k, in case

$$\min_j \sum_{r=1}^{h-1} p_{k-r+1, j} \geq \max_j \sum_{r=1}^{h-1} p_{k-r, j} \quad \ldots\ldots (*)$$

whenever $2 \leq h \leq k$

Lemma 1.1.1 If BCD conditions hold for machine M_k, then for any sequence $\bar{p} = (j_1 \ldots j_n)$ the time required to process all the jobs on machines $M_1, M_2, \ldots, M_k$, is given by

$$t(\bar{p}, k) = \sum_{r=1}^{k-1} p_{rj_1} + \sum_{s=1}^{n} p_{kj_s}$$

Proof : Let C_{kj} be the completion time of job j on machine M_k while using $\bar{p}$. To prove the lemma it is sufficient to show that

$$C_{k\, j_{s-1}} \geq C_{k-1, j_s}, \quad 2 \leq s \leq n.$$

But $$C_{k-1, j_s} = \text{Max}[C_{k-2, j_s}, C_{k-1, j_{s-1}}] + p_{k-1, j_s} \quad \ldots.1.1.3$$

Expanding the R.H.S. of 1.1.3 above by repeated use of the same relationship, we get

$$C_{k-1, j_s} = \max_{1 \leq q \leq k-1} [C_{q\, j_{s-1}} + \sum_{r=q}^{k-1} p_{rj_s}] \quad \ldots.1.1.4$$

Thus we have to show that each term in R.H.S. of 1.1.4 is less than or equal to $C_{k\, j_{s-1}}$.

Consider the term for $q = k-1$, viz., $C_{k-1, j_{s-1}} + p_{k-1, j_s}$

Now,

$$C_{kj_{s-1}} = \text{Max}[C_{k-1,j_{s-1}},\ C_{k\ j_{s-2}}] + p_{kj_{s-1}}$$

$$\geq C_{k-1,j_{s-1}} + p_{k-1,j_s}$$

since by the first BCD condition

$$\min_j p_{kj} \geq \max_j p_{k-1,j}$$

Consider the term for $q = k-2$, namely, $C_{k-2,\ j_{s-1}} + p_{k-1,j_s} + p_{k-2,j_s}$

We have

$$C_{kj_{s-1}} \geq C_{k-2,j_{s-1}} + p_{kj_{s-1}} + p_{k-1,j_{s-1}}$$

by definition and

$$p_{k,j_{s-1}} + p_{k-1,j_{s-1}} \geq p_{k-1,j_s} + p_{k-2,j_s}$$ from BCD condition for $h = 3$.

Hence,

$$C_{kj_{s-1}} \geq C_{k-2,j_{s-1}} + p_{k-1,j_s} + p_{k-2,j_s}\ .$$

Proceeding in the same lines, we see that

$$C_{kj_{s-1}} \geq C_{k-1,j_s} \quad \forall \quad 2 \leq s \leq n.$$

Thus machine M_k does not wait for any of the jobs $j_2 \ldots j_n$. But, anyhow, unless the first job j_1 is done on all the

machines up to M_{k-1}, machine M_k cannot start processing the first job. Thus the idle time on machine M_k is $\sum_{r=1}^{k-1} p_{rj_1}$. This proves the lemma.

The following theorem follows immediately from Lemma 1.1.1.

Theorem 1.1.1 : If there exists a k such that BCD conditions hold for k^{th} machine then the corresponding $(n/m/F/F_{max})$ problem reduces to that of solving n, $(n/m-k+1/F/F_{max})$ problems involving the last $m-k+1$ machines with the restriction that for $1 \leq j \leq n$, in the j^{th} problem, job j is fixed to be the first job. An optimal sequence for the problem is obtained by selecting the 'best' (j_o, S_{j_o}) sequence where S_j is an optimal sequence of the $n-1$ jobs other than job j, for the j^{th} problem.

Definition 1.1.2 : Forward Cumulative Dominance (FCD). Similar to BCD, we define FCD conditions for machine M_k as

$$\min_j \sum_{r=1}^{h} p_{k+r-1,j} \geq \max_j \sum_{r=1}^{h} p_{k+r,j} \qquad \ldots(**)$$

$$\forall\ 1 \leq h \leq m-k.$$

Lemma 1.1.2 : If FCD conditions hold for machine M_k, then for any partial sequence $\sigma = (j_1 \ldots j_q)$ and for any $j \in N - \sigma$ we have

$$t(\sigma j, r) = t(\sigma j, r-1) + p_{rj}$$

for all r such that $k+1 \leq r \leq m$.

Proof : The result is true for $\sigma = \phi$. Now we proceed to prove the result by induction on the number of jobs in σ. Assuming the result for any σ with $q \leq s$, we prove the result for any σ with $q = s + 1$. So let σ be any partial sequence $(j_1 \cdots j_{s+1})$.

Now we observe,

$$t(\sigma j,\ k+1) = \text{Max}[t(\sigma j,\ k),\ t(\sigma,\ k+1)] + p_{k+1,j}$$

$$= \text{Max}[t(\sigma j,\ k),\ t(\sigma,k)+p_{k+1,j_{s+1}}]+p_{k+1,j} .$$

But $t(\sigma j,\ k) = \text{Max}[t(\sigma j,\ k-1),\ t(\sigma,k)] + p_{kj}$,

so $t(\sigma j,k) \geq t(\sigma,k) + p_{kj} \geq t(\sigma,k) + p_{k+1,j_{s+1}}$

by FCD condition (**) for h = 1

Hence $t(\sigma j,\ k+1) = t(\sigma j,\ k) + p_{k+1,j}$

We now use induction on r.

Assuming the equality to be true for all r such that $k+1 \leq r \leq v$, we prove the result for $v+1$

$$t(\sigma j,\ v+1) = \text{Max}\ [t(\sigma j,\ v),\ t(\sigma,\ v+1)] + p_{v+1,j}$$

But

$$t(\sigma j,\ v) \geq t(\sigma,\ k) + \sum_{r=k}^{v} p_{rj}$$

by the induction hypothesis on r

and $t(\sigma, v+1) = t(\sigma,k) + \sum_{r=k+1}^{v+1} p_{rj_{s+1}}$

by the induction hypothesis on v

From these two we get using the FCD condition (**) for $h = v + 1 - k$, $t(\sigma j, v+1) = t(\sigma j, v) + p_{v+1,j}$.

This completes the proof of the lemma.

We have the following theorem when FCD conditions hold for machine M_k. The proof of this theorem is immediate from Lemma 1.1.2.

Theorem 1.1.2 : If there exists a k such that FCD conditions hold for machine M_k, then the corresponding $(n/m/F/F_{max})$ problem reduces to that of solving n, $(n/k/F/F_{max})$ problems involving the first k machines with the restriction that for $1 \leq j \leq n$, in the j^{th} problem the last job is fixed to be j. An optimal sequence for the problem is obtained by selecting the 'best' (S_{j_o}, j_o) sequence where S_j is an optimal sequence of the n-1 jobs other than j, for the j^{th} problem.

Remarks

1. Note that when BCD or FCD conditions hold for machine M_k, each of the n problems considered in theorems 1.1.2 and 1.1.2 is in essence a problem with n-1 jobs. Further the reduction in number of machines will certainly reduce the

number of $L_r(\sigma)$, to be worked out for each partial σ while using Branch and Bound method, Lomnicki and Brown (1966). Of course the additional restriction that either $j_1 = j$ or $j_n = j$ can be incorporated easily in the Branch and Bound method.

2. When the problem size reduces considerably efficient methods to solve may exist, for example, when m is 1 or 2. This is a great advantage in reducing the computational effort required.

3. The above theorems 1.1.1 and 1.1.2 hold good for more general objective functions as well. A study of some interesting special cases is done in the next chapter.

4. The above theorems are true, even if the no passing restriction is relaxed. Only S_j need to be taken as an optimal sequence matrix, which gives one sequence for each of the machines, for the corresponding j^{th} problem and (j_o, S_{j_o}) or (S_{j_o}, j_o) need to be similarly interpreted.

Now going back to the special cases I to V considered earlier in this section, we note Case I $\Rightarrow$ For machine M_1 i.e., k = 1, FCD condition (**) holds for every h, $1 \leq h \leq m-k-1 = m-2$.

Proposition 1.1.1 : If for machine M_1, FCD condition (**) holds for all h such that $1 \leq h \leq m-2$, then for any sequence $\bar{p} = (j_1 \ldots j_n)$, the idle time on the last machine is given by

$$g(\bar{p}) = \operatorname*{Max}_{1 \leq u \leq n} \left\{ \sum_{s=1}^{u} \left[\sum_{r=1}^{m-1} p_{rj_s} \right] - \sum_{s=1}^{u-1} \left[\sum_{r=2}^{m} p_{rj_s} \right] \right\}$$

Proof : Let X_i be the idle time immediately before job j_i on machine M_m.

We have, $X_1 = \sum_{r=1}^{m-1} p_{rj_1}$

$$X_i = \operatorname{Max}\left[t((j_1 \ldots j_i), m-1) - \sum_{s=1}^{i-1} X_s - \sum_{s=1}^{i-1} p_{mj_s}, 0 \right]$$

But from Lemma 1.1.2, with $k = 1$ and $1 \leq h \leq m-2$

$$t((j_1 \ldots j_i), m-1) = t((j_1 \ldots j_i), m-2) + p_{m-1, j_i}$$

$$= \sum_{s=1}^{i} p_{1j_s} + \sum_{r=2}^{m-1} p_{rj_i}$$

We get this by recursively substituting in the R.H.S. of the above equation for $t((j_1 \ldots j_i), m-1)$.

Therefore $X_i = \operatorname{Max}\left[\sum_{s=1}^{i} p_{1j_s} + \sum_{r=2}^{m-1} p_{rj_i} - \sum_{s=1}^{i-1} p_{mj_s} - \sum_{s=1}^{i-1} X_s, 0 \right]$

Now it is immediate that $g(\bar{p}) = \sum_{u=1}^{n} X_u$ is as given by the proposition.

We note that $g(\bar{p})$ is same as under case I. Hence the same procedure can be used to solve the problem when FCD condition (**) holds for machine M_1 for $1 \leq h \leq m-2$. It is easy to see that case II implies that for machine M_m, BCD condition(*) holds for all h such that $2 \leq h \leq m-1$.

Proposition 1.1.2 : If for machine M_m BCD condition (*) holds for all h such that $2 \leq h \leq m-1$, then for any sequence $\bar{p} = (j_1 \ldots j_n)$, the idle time on the last machine, $g(\bar{p})$, is as given in proposition 1.1.1.

Proof : Let $X_{r,i}$ be the idle time on the machine M_r, immediately before job j_i is done, while using the sequence $\bar{p} = (j_1 \ldots j_n)$. Let r_o be the largest r such that $X_{r_o,i}$ is zero and $r_o \geq 2$.

Case a : There exists such an r_o. We are given

$$\min_j \sum_{r=1}^{h-1} p_{m-r+1,j} \geq \max_j \sum_{r=1}^{h-1} p_{m-r,j}$$

whenever $2 \leq h \leq m-1$

Now

$$t((j_1 \ldots j_i), m-1) = t((j_1 \ldots j_{i-1}), r_o) + \sum_{r=r_o}^{m-1} p_{r\, j_i}$$

since $X_{ri} > 0$ for all $r \geq r_o \geq 2$

But, $\sum_{r=r_o}^{m-1} p_{rj_i} \leq \sum_{r=r_o+1}^{m} p_{rj_{i-1}}$ from the BCD

Condition (*) given above corresponding to $h = m-r_o+1$.

Therefore,

$$t((j_1 \ldots j_{i-1}),\ r_o) + \sum_{r=r_o}^{m-1} p_{rj_i} \leq t((j_1 \ldots j_{i-1}), r_o) + \sum_{r=r_o+1}^{m} p_{rj_{i-1}}$$

$$\leq t((j_1 \ldots j_{i-1}), m)$$

by definition of $t((j_1 \ldots j_{i-1}), m)$.

Thus $X_i = X_{m,i} = 0$.

Case b : There does not exist such an r_o i.e.,

$$X_{r,1} > 0 \quad \text{for} \quad r \geq 2.$$

But $X_{1,i} = 0$ for all i. Therefore,

$$t((j_1 \ldots j_i),\ m-1) = \sum_{s=1}^{i} p_{1j_s} + \sum_{r=2}^{m-1} p_{r\ j_i}, \quad \text{since } X_{ri} > 0$$

for all $r \geq 2$.

Therefore $X_i = \text{Max}\left[\sum_{s=1}^{i} p_{1j_s} + \sum_{r=2}^{m-1} p_{rj_i} - \sum_{s=1}^{i-1} p_{mj_s} - \sum_{s=1}^{i-1} X_s, 0 \right]$

Combining both case 'a' and case 'b' we immediately obtain the required result.

Case III $\Rightarrow$ For machine M_{m-1}, BCD conditions hold.

Case IV $\Rightarrow$ For machine M_k, $k = 2$, FCD conditions hold.

Case V $\Rightarrow$ For some machine M_k, both, BCD conditions and FCD conditions hold.

Thus, the conditions given in cases I to V can be weakened using these implied CD-conditions.

Remark : We note that the following conditions (a) to (e) also imply the same CD-conditions as implied by cases I to V, respectively. Hence similar procedures are applicable.

(a) $\min_j p_{rj} \geq \max_j p_{r+1,j}$ for all r, $1 \leq r \leq m-2$.

(b) $\min_j p_{rj} \geq \max_j p_{r-1,j}$ for all r, $3 \leq r \leq m$.

(c) $\min_j p_{rj} \geq \max_j p_{r-1,j}$ for all r, $2 \leq r \leq m-1$.

(d) $\min_j p_{rj} \geq \max_j p_{r+1,j}$ for all r, $2 \leq r \leq m-1$.

(e) for some $2 \leq k \leq m-1$, we have

both $\min_j p_{rj} \geq \max_j p_{r-1,j}$ for all r, $2 \leq r \leq k$,

and $\min_j p_{rj} \geq \max_j p_{r+1,j}$ for all r,

$k \leq r \leq m-1$.

1.2 An extension of the two stage sequencing problem $(n/(m,1)/F/F_{max})$.

1.2.0 Introduction. The problem considered in this section is $(n/(m,1)/F/F_{max})$ problem with usual assumptions A-1 through A-8 excepting assumption A-7. This happens to be an extension of Johnson's 2-stage and n jobs problem. There are m 'identical' machines of type A in stage 1 and one machine of type B in stage 2. A Branch and Bound method is developed by Arthanari and Ramamurthy (1971), for finding a 'plan' which minimises the total elapsed time in processing all the jobs. It has been shown that the minimum elapsed time is attained at least for one member of the set of schedules called preferred schedules. First, we give the results required for the Branch and Bound algorithm and present the algorithm. A heuristic rule is then proposed which has been found to be nearly optimal. In the light of this nearly optimal rule, a modified algorithm is suggested.

Bagga and Mittal (1973) claim that they have developed a simpler algorithm for the problem with 2 machines of type A and

1 of type B. K.G.Ramamurthy (1973) in an unpublished note, has given counter examples to show that their claims are incorrect and has also given a correct necessary and sufficient conditions for optimality of a partition of n jobs into two sets, so as to minimisè the total elapsed time, in a single stage two identical machines problem. We briefly discuss this at the end.

1.2.1 Notations and statement of the problem

Since there are only two stages, we deviate from the usual notation for m stage problem and use as in Johnson's paper(1954). Here we denote by A_j and B_j the processing time for job j on a machine of type A and that on the machine of type B respectively.

The optimization problem considered is to find a plan which gives a schedule for processing jobs on machines of type A and a sequence for processing jobs on the machine of type B, satisfying the assumptions made, so as to yield a minimum total elapsed time. We assume $n > m$, as otherwise the problem is trivial.

Definition 1.2.1 A schedule $\alpha = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ specifies $\sigma_i = (j^i_1, \ldots, j^i_{k_i})$, $i = 1,2,\ldots,m$ where j^i_r represents the r^{th} job to be taken up on the i^{th} machine of type A.

Definition 1.2.2 A schedule α is said to be partial(complete) in case $\sum_{i=1}^{m} k_i < n(=n)$. Let $\underline{C}$ denote the set of all complete schedules and $\underline{P}$ the set of all partial schedules.

Definition 1.2.3 For any $\alpha \in \underline{P}(\_)\underline{C}$, we define T_α to be the minimum elapsed time possible among the complete schedules whose initial part is α.

Therefore the problem is equivalent to finding α^* that minimizes T_α over $\alpha \in \underline{C}$.

Definition 1.2.4 Let $t(\alpha) = \underset{1 \le i \le m}{\text{Min}} \sum_{j \in \sigma_i} A_j$ with convention $\sum_{j \in \phi} A_j = 0$. Let f_α for $\alpha \in \underline{P}(\_)\underline{C}$, be the vector

$f_\alpha = (f_1, f_2, \ldots, f_n)$

where $f_j = \begin{cases} \text{Completion time of job } j \text{ on a machine of stage } A, \text{ if } j \in \alpha \\ t(\alpha) + A_j \quad \text{if } j \notin \alpha \end{cases}$

For $f \in R^n$, $f \ge 0$, we define J_f to be the vector $(j_1, j_2, \ldots, j_n)$ such that $f_{j_1} \le f_{j_2} \le \ldots \le f_{j_n}$

Definition 1.2.5 For a $f \ge 0$, we define $T(f)$ to be the minimum elapsed time possible under the constraint imposed by f, i.e., taking f_j as the completion time of job j on the stage A.

Definition 1.2.6 A schedule $\alpha \in \underline{P}(\_)\underline{C}$ is called a preferred schedule if the set

$$D = \{ j \in N / f_j - A_j > t(\alpha) \} \text{ is empty}$$

We denote the set of all preferred complete schedules by $\underline{C}^*$ and the set of all preferred partial schedules by $\underline{P}^*$.

1.2.2 Main Results and the algorithm.

Lemma 1.2.1 Given any f such that $f \in R^n$, $f \geq 0$, $T(f)$ is attained when jobs and processed according to the sequence J_f on the machine of type B and its value is given by

$$T(f) = \sum_{j \in N} B_j + \max_{1 \leq r \leq n} \left(f_{j_r} - \sum_{u=1}^{r-1} B_{j_u} \right)$$

The proof of this lemma is based on a reasoning similar to that used in Johnson (1954) and so is omitted.

Lemma 1.2.2 If $f^* \leq f$ then $T(f^*) \leq T(f)$.

Proof. Observe that

$$T(f) \geq \sum_{j \in N} B_j + \max_{1 \leq r \leq n} \left(f^*_{j_r} - \sum_{u=1}^{r-1} B_{j_u} \right)$$

since $f^*_j \leq f_j$ for every j.

The R.H.S. expression is the elapsed time corresponding to the sequence J_f on the machine of type B when f* gives the completion times for stage A and so $\geq T(f^*)$. This lemma shows that we need only consider schedules $\alpha \in \underline{C}$ for which (1) there is no idle time between any two suceessive jobs processed on any machine of type A, and (2) there is no idle time between time 0 and the start of the first job on any machine of type A.

Lemma 1.2.3 There exists a preferred schedule that is optimal.

Proof : If $\alpha \in \underline{C}$, then the only constraint imposed by α on processing of the jobs on the machine of type B is on the earliest instants of time at which each job can be taken up for processing and this is given by f_α. By definition of T_α and $T(f_\alpha)$ it follows

$$T_\alpha = T(f_\alpha) \quad \text{if} \quad \alpha \in \underline{C}$$

Since $\underline{C}^* \subseteq \underline{C}$, if we prove that for any $\alpha \in \underline{C}$, there exists at least one $\alpha^* \in \underline{C}^*$ such that $f_{\alpha^*} \leq f_\alpha$ then the required result follows from Lemma 1.2.2. Let $\alpha \in \underline{C}$ be any schedule which is not a preferred schedule.

Let the $\underset{1 \leq i \leq m}{\text{Min}} \left(\sum_{j \in \sigma_i} A_j \right) = t(\alpha)$ be attained for the h^{th} machine. Since α is not a preferred schedule, we have

$$D = \{ j \in N / f_j - A_j > t(\alpha) \} \neq \phi.$$

Let $f_r - A_r = \underset{j \in D}{\text{Max}} (f_j - A_j)$. Then allot r^{th} job to h^{th} machine to get a new schedule α' and the number of elements in D will decrease at least by one. It is obvious that $f' \leq f$ since $f'_j = f_j$ for all $j \in N$, $j \neq r$ and $f'_r = t(\alpha) + A_r < f_r$. Since D is finite it is possible to reach a preferred schedule α^* by repeating the above process and $f_{\alpha^*} \leq f_\alpha$. This completes the proof of the lemma.

Lemma 1.2.4 If $\alpha \in P$ then $T_\alpha \geq T(f_\alpha)$.

Proof : Let $\alpha \in P$ and let α' be a complete schedule whose initial part is α and such that $T_\alpha = T_{\alpha'}$.

Since in schedule α', the jobs $j \in \alpha$ always precede jobs $j \in N - \alpha$, we have

$$f'_j = f_j \quad \text{for} \quad j \in \alpha$$

$$\geq A_j + t(\alpha) = f_j \quad \text{for} \quad j \in N - \alpha.$$

Therefore,

$$T_\alpha = T_{\alpha'} = T(f_{\alpha'}) \quad \text{since} \quad \alpha' \in G$$

and $$T(f_{\alpha'}) \geq T(f_\alpha) \quad \text{since} \quad f_{\alpha'} \geq f_\alpha.$$

Lemmas 1.2.1 and 1.2.4 give us a simple method of calculating lower bounds of partial schedules and actual minimum elapsed time when $\alpha \in G$.

A Branch and Bound method using these results is given below.

The Algorithm

We represent the preferred schedules in the form of nodes of a tree. A node representing a schedule belonging to $\underline{P}^*(\_)\underline{C}^*$ is said to be of q^{th} order if the cardinality of the set of jobs considered by the schedule is q. The node of order zero is denoted by $(.)$. All nodes of order greater than zero are denoted by the corresponding α. A node of order of q $(0 \leq q \leq n-1)$ is said to be 'active' at any stage of computation if it has no descendants in the partial tree generated so far and we have not yet proved conclusively that this partial schedule cannot eventually give us a complete schedule better than the best one known so far. At any stage of computation let π denote the set of all active nodes. Let $T(\alpha^*)$ and α^* represent the minimum total elapsed time obtained so far, for a complete schedule and the corresponding schedule respectively. Initially we set

$$\pi = \{(.)\}, \quad T(\alpha^*) = \infty, \quad \alpha^* = \text{Dummy}.$$

Step 0 : Choose node $(.)$ for branching and enter step 1.

Step 1 : Let the order of the node selected for branching be q. Designate this node by α and delete this node from π.

If $q = 0$, go to step 2. Otherwise go to step 3.

Step 2 : Generate $\lambda = n_{c_m}$ descendants of α using the following branching rule.

Branching Rule 1 : Each one of these descendants represents the set of first m jobs to be taken up for processing on machines of type A. The actual allocation of the selected m jobs to m machines is arbitrary and any one of the $m!$ possibilities will be sufficient. Let α^i be the i^{th} descendant of α and the order of each one of them is m. Set $q = m$ and go to step 4.

Step 3 : Generate $\lambda = (n-q)$ descendants of α using the following branching rule.

Branching Rule 2 : Find $t(\alpha)$ and suppose the minimum is attained for the h^{th} machine and the number of jobs on h^{th} machine in schedule α is k_h. Allot jobs $j \in N - \alpha$ to the h^{th} machine systematically one by one for the $(k_h+1)^{th}$ position to get all the descendants of α. The number of descendants will be $n - q$ and each one of them will be of order $q + 1$. Denote them by $\alpha^1, \ldots, \alpha^\lambda$, set $q = q + 1$ for all these nodes and go to step 4.

Step 4 : Calculate the lower bound $T(f_{\alpha^i})$ for $i = 1, 2, \ldots, \lambda$ as follows : For each α^i the vector f_{α^i} is obtained as per

definition of f_{α^i}. $J_{f_{\alpha^i}}$ is found. If $J_{f_{\alpha^i}} = (j_1 \ldots j_n)$ then calculate Z_{j_n} using the formula

$$Z_{j_1} = f_{j_1}^{\alpha^i} + B_{j_1}$$

and $$Z_{j_r} = \text{Max}(Z_{j_{r-1}}, f_{j_r}^{\alpha^i}) + B_{j_r}, \quad 2 \leq r \leq n$$

we have $T(f_{\alpha^i}) = Z_{j_n}$. If $n - q = 1$ go to step 5, otherwise go to step 6.

Step 5 : If $T(f_{\alpha^1}) < T(\alpha^*)$, put $T(\alpha^*) = T(f_{\alpha^1})$ and $\alpha^* = \alpha^1$. Delete all nodes with lower bound greater than $T(\alpha^*)$ from π and go to step 8. Otherwise go to step 8 directly.

Step 6 : Let $R = \{ \alpha^i / T(f_{\alpha^i}) < T(\alpha^*) \}$

If $R = \phi$ go to step 8. Otherwise, set $\pi = \pi(\_)R$ and go to step 7.

Step 7 : Find α such that $T(f_\alpha) = \underset{\alpha^i \varepsilon R}{\text{Min}} [T(f_{\alpha^i})]$. Use any tie breaking rule, in case of ties. Use the node α for branching and go to step 3.

Step 8 : If $\pi = \phi$ go to step 10. Otherwise, go to step 9.

Step 9 : Choose a node of highest order from the set π, with smallest lower bound. Designate this node as α and use this α for branching. Go to step 3.

Step 10 : We have obtained an optimal schedule α^* with $T(\alpha^*)$ as minimum total elapsed time. For the second stage we use $J_{f_{\alpha^*}}$ as the optimal sequence. Stop.

Example 1.2.1 Consider a $(4/(2,1)/F/F_{max})$ problem with the data given below.

j	A_j	B_j
1	5	2
2	18	3
3	12	1
4	14	3

The tree obtained by using the algorithm is given in Fig. 1.2.1. The optimal total elapsed time for this problem is 27. The optimal schedule for machines of type A is $\alpha^* = (1,2\ ;\ 4,3)$ with $f_{\alpha^*} = (5,23,26,14)$. Hence an optimal sequence for the machine of type B is given by $J_{f_{\alpha^*}} = (1,4,2,3)$. We illustrate the calculation of $T(\alpha)$ for $\alpha = (1;4)$.

Now $t(\alpha) = 5$ is attained for machine $h = 1$. So $f_\alpha = (5,\ 23,\ 17,\ 14)$. Then

$$J_{f_\alpha} = (1,\ 4,\ 3,\ 2).$$

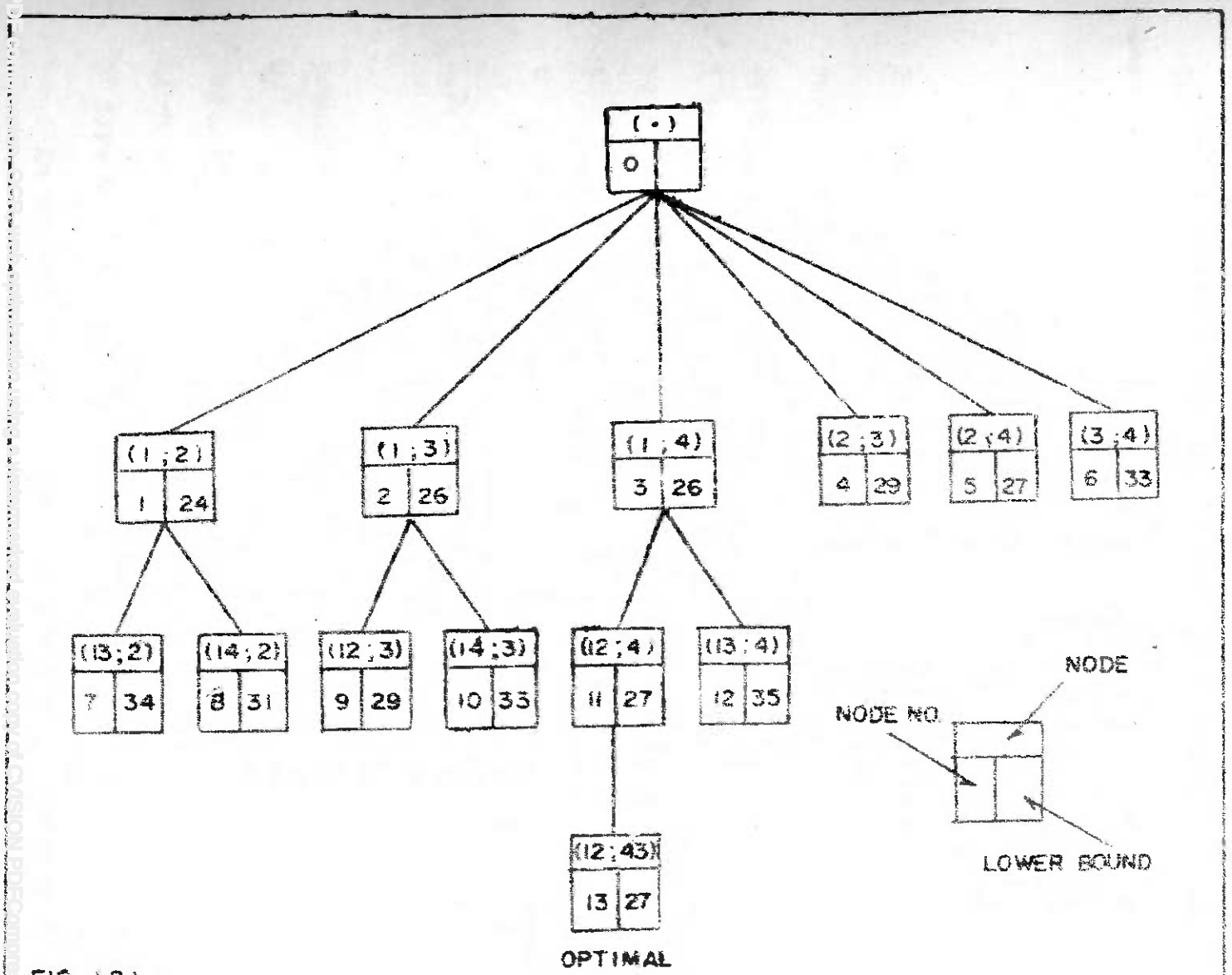

(•)
0
(1 ; 2)
1 24
(1 ; 3)
2 26
(1 ; 4)
3 26
(2 ; 3)
4 29
(2 ; 4)
5 27
(3 ; 4)
6 33
(13;2)
7 34
(14;2)
8 31
(12;3)
9 29
(14;3)
10 33
(12;4)
11 27
(13;4)
12 35
(12;43)
13 27
OPTIMAL
NODE
NODE NO.
LOWER BOUND

FIG. 1.2.1

Therefore $Z_1 = f_1 + B_1 = 7$

$Z_4 = \text{Max}(f_4, Z_1) + B_4 = 17$

$Z_3 = \text{Max}(f_3, Z_4) + B_3 = 18$

and $Z_2 = \text{Max}(f_2, Z_3) + B_2 = 26$

Thus $T(f_\alpha) = Z_{j_n} = Z_2 = 26.$

For $\alpha = (1\,;\,4)$ as $t(\alpha)$ is attained for $h = 1$ while branching from α we fix systematically jobs 2 and 3 after 1 in machine 1 of type A. In the first branching from (.) we generate 4_{C_2} descendants of (.) and calculate the bounds as illustrated for (1;4). As the lower bound for (1;2) is the lowest we choose (1;2) for branching and so on.

1.2.3 A Nearly Optimal Rule

With stages A and B solve the corresponding Johnson's problem. Let $(s_1, s_2, \ldots, s_n)$ be an optimal sequence thus obtained. We consider the schedule in which $s_1, \ldots, s_m$ are allotted to the machines of type A at time 0 and after $s_1, \ldots, s_i$ being allotted to machines of type A, s_{i+1} is allotted to a machine of type A such that it is completed as early as possible.

An experimental study of 174 randomly generated problems with different distributions and sizes, was done to find the efficiency of this rule. The results are given below.

A set of 50 $(4/(2,1)/F/F_{max})$ problems was generated randomly from each of N(50, 10), N(50, 20) and 'Uniform (0-99)'. The randomly generated values were rounded off to the nearest integer. A fourth set of 24 problems with n ranging from 5 to 20 was also generated randomly from 'Uniform (0-99)'. The following table gives the details about the results obtained.

Problem set	No. of problems solved	No. of problems in which optimal solution was obtained by the near optimal rule	No. of problems in which the solution was within 5 per cent of the optimal value	Max'/. deviation from optimal value
I	50	42	49	10.64
II	50	37	47	8.80
III	50	34	44	9.80
IV	24	22	24	3.07
Total	174	135	164	
'/.	100.00	77.58	94.25	

Thus 94.25 per cent of the problems yielded nearly optimal solutions.

Next we test whether the proportion of problems in which the near optimal rule gives an optimal solution is the same in all the populations considered above.

Let $H_o : p_1 = p_2 = p_3 = p_4$ where p_i is the proportion of problems in the i^{th} set for which optimal solution was found using the Near optimal rule.

We use a χ^2-test to test this null hypothesis.

$n = 174;\ n_1 = n_2 = n_3 = 50\ ;\ n_4 = 24$

$x = 135;\ x_1 = 42;\ x_2 = 37;\ x_3 = 34;\ x_4 = 22.$

Statistic $u = \frac{n^2}{x(n-x)} \left[\sum_{i=1}^{4} \frac{x_i^2}{n_i} - \frac{x^2}{n} \right]$

$$= \frac{30276}{135 \times 39} \left[\frac{4289}{50} + \frac{484}{24} - \frac{18225}{174} \right]$$

$$= 6.9575$$

Under H_o, u follows a χ^2-distribution with 3 degrees of freedom and the upper 5 percent point of χ_3^2 is 7.81. Since the observed value of u is less than 7.81, we conclude that these proportions are not significantly different.

Similarly the proportions corresponding to problems for which the solutions were found to be within 5 percent of the optimum were tested for equality. They are not significantly different (Observed u = 7.39).

Modified Algorithm

In the light of the above experimental study we suggest the following modification in the algorithm given. We initialize

in step 0, by taking α^* as the nearly optimal sequence obtained using the rule given above. This will result in deleting many nodes from π, as α^* is nearly optimal.

1.2.4 Comments on a paper by Bagga and Mittal

Bagga and Mittal (1973) proposed the following method for solving the $(n/(2,1)/F/F_{max})$ problem.

Step 1 : Partition N into subsets N_1 and N_2 such that $\text{Max}[\alpha(N_1), \alpha(N_2)]$ is minimised over all possible partitions of N, where

$$\alpha(D) = \sum_{j \in D} A_j \quad \text{if} \quad \phi \neq D \subseteq N$$

$$= 0 \quad \text{if} \quad D = \phi.$$

Let N_1^*, N_2^* be such a partition of N.

Step 2 : Find a permutation σ_1 of N_1^* and a permutation σ_2 of N_2^* such that the schedule, $\sigma = (\sigma_1, \sigma_2)$ minimises the total elapsed time for processing the jobs on the two stages. This gives an optimal solution to the problem.

They suggest a procedure for partitioning N into two subsets in an optimal way as required in step 1, which is found to be incorrect. Even if we find the optimal partition in step 1, there is no guarentee of obtaining an optimal solution

to the problem as it is not sufficient to consider only the $(n_1^*!)\,(n_2^*!)$ schedules, where n_i^* = the number of jobs in N_i^*.

To establish that their method does not lead to an optimal solution to the problem, the following counter example is given in an unpublished note by Ramamurthy (1973).

Example 1.2.2

Job j	A_j	B_j
1	6	4
2	8	3
3	7	1
4	7	1

Let $N_1^* = \{1,2\}$ and $N_2^* = \{3,4\}$ we have

$$\alpha(N_1^*) = \alpha(N_2^*) = 14 \quad \text{and}$$

$$14 \leq \text{Max}(\alpha(N_1), \alpha(N_2))$$

for any partition N_1, N_2 of N. The following table gives the details regarding the $2! \times 2! = 4$ schedule to be considered in step 2.

Sl. no.	Schedule	Completion time of the job								T.E.
		On stage A				On stage B				
		1	2	3	4	1	2	3	4	
1	(1,2 / 3,4)	6	14	7	14	10	17	11	18	18
2	(1,2 / 4,3)	6	14	14	7	10	17	18	11	18
3	(2,1 / 3,4)	14	8	7	14	18	11	8	19	19
4	(2,1 / 4,3)	14	8	14	7	18	11	19	8	19

Thus schedules 1 and 2 are optimal according to Bagga and Mittal's method. But for the schedule (1,3/2,4) we have actually the total elapsed time = 16. However, it is seen for this schedule

$$\text{Max}[\alpha(N_1), \alpha(N_2)] = 15$$ which is strictly greater than the optimal value 14.

Bagga and Mittal assert that a partition N_1, N_2 is optimal if it satisfies the condition

$$|\alpha(N_1) - \alpha(N_2)| < \text{Min } [A_j, j \in N].$$

This is disproved by the following counter example.

Example 1.2.3

Consider the data given below.

j	1	2	3	4	5	6	7	8
A_j	7.0	11.5	9.0	3.0	5.0	6.0	2.0	4.0

Let $N_1 = \{ 3, 4, 5, 6 \}$ and $N_2 = \{1, 2, 7, 8\}$. We have $\alpha(N_1) = 23.0$ and $\alpha(N_2) = 24.5$.

$$|\alpha(N_1) - \alpha(N_2)| = 1.5 < 2 = \text{Min } [A_j, j \in N].$$

Hence N_1, N_2 should be an optimal partition according to Bagga and Mittal (1973). But for the partition,

$$N_1' = \{ 1, 3, 4, 5 \} \quad \text{and} \quad N_2' = \{ 2, 6, 7, 8 \}$$

we have

$$\alpha(N_1') = 24.0, \quad \alpha(N_2') = 23.5$$

and $$|\alpha(N_1') - \alpha(N_2')| = 0.5 < 1.5.$$

Hence the condition given by them is not a sufficient condition. Ramamurthy(1973) gives a necessary and sufficient condition for a partition to be optimal.

1.3 Some Observation on Switching Rules

1.3.0 Introduction

Smith (1956) has given a sufficient condition on the objective function for contiguous binary switching (CBS) to yield an optimal solution. Elmaghraby (1971) defines convex graphs and shows the equivalence of Smith's theorem with the

preference graph corresponding to the problem being convex. John G. Rau (1971) has used CBS rule to find an optimal permutation that minimizes a function under certain conditions. In section 1.3.1 we extend the definition of preference graph of CBS rule as given by Elmaghraby to general switching rules. Next we show the equivalence of Rau's conditions to some simpler ones and in section 1.3.3 we give a 'good' algorithm for cases when CBS rule is applicable but transitivity does not hold for the function $g(\cdot,\cdot)$ referred to in Smith's (1956) theorem.

1.3.1 Convex Graphs and Switching Rules

We now extend the definition of preference graph of CBS rule as given by Elmaghraby (1971) to general switching rules.

Definition 1.3.1 : A switching rule S is a function defined on $\underline{Q}$, the set of permutations of $(1\ 2 \ldots n)$ into the collection of all subsets of $\underline{Q}$.

$$S : \underline{Q} \rightarrow 2^{\underline{Q}} \quad \text{such that, for every } x \in \underline{Q}$$

$$S(x) \neq \phi.$$

Definition 1.3.2 : $x, y \in \underline{Q}$ are said to be adjacent if

$$y \in S(x) \quad \text{and} \quad x \in S(y).$$

Let $x, y \in \underline{Q}$. x is said to be connected to y, if there exist $x_1, \ldots, x_k \in \underline{Q}$ such that $x_1 \in S(x)$; $x_i \in S(x_{i-1})$ for all, $2 \leq i \leq k$ and $y \in S(x_k)$.

In general x is connected to y does not imply y is connected to x.

A switching rule S is said to be symmetric if

$x \in S(y) \Rightarrow y \in S(x)$ for $x, y \in \underline{\underline{Q}}$.

A switching rule S is said to be connected if

$\forall$ x, y $\in \underline{\underline{Q}}$, x is connected to y.

Observe that, when a symmetric switching rule is used, if x is connected to y, then y is connected to x.

Let f be a real valued function defined on $\underline{\underline{Q}}$, to be minimised over $\underline{\underline{Q}}$. Consider a symmetric, connected switching rule S.

Let $A = \{(x,y) : x, y \in \underline{\underline{Q}}$ and x and y are adjacent$\}$.

Let $\Gamma'_{(x)} = \{(x, y) \in A : f(y) < f(x)\}$

$\Gamma''_{(x)} = \{(x, y) \in A : f(y) = f(x)\}$

$G_S = (\underline{\underline{Q}}, A)$ is a connected graph.

Now $\forall$ $(x, y) \in \Gamma'_{(x)}$ introduce the direction from x to y, $\forall$ $x \in \underline{\underline{Q}}$. The resultant graph G(S, f) is called the preference graph corresponding to the function f and the switching rule S.

Definition 1.3.3 : $G(S, f)$ is convex iff the set of all nodes of $G(S, f)$, namely $\underline{\underline{Q}}$ can be partitioned into two sets B and R such that

(i) $B \neq \phi \Rightarrow \forall\, x \in B,\ \Gamma'_{(x)} \neq \phi$

(ii) $R \neq \phi$

and (iii) $f(x) = \text{constant for all } x \in R.$

Suppose for some switching rule S and a function f, the corresponding preference graph $G(S, f)$ is convex, we have the following simple procedure to get an optimal solution:

> Start with any $x \in \underline{\underline{Q}}$. If $\Gamma'_{(x)}$ is empty, then x is optimal. Otherwise, choose any $y \in \Gamma'_{(x)}$ and repeat the process. It is easy to see that this procedure leads to an optimal solution, if $G(S, f)$ is convex.

Of course switching rules are of interest only when their use reduces the computation required, For example S given by, $\forall\, x \in \underline{\underline{Q}},\ S(x) = \underline{\underline{Q}}$ is a switching rule that is symmetric, and connected and for any real valued function f on $\underline{\underline{Q}}$, the corresponding graph $G(S, f)$ is convex. But this only means, one needs to do complete enumeration of all the permutations. On the otherhand if for some f, CBS rule gives a convex graph,

the procedure described above with the CBS rule gives a simple method to get an optimal sequence. Actually a better procedure can be given in this case (see section 1.3.4).

Another simple switching rule of interest is the Binary Switching rule (BS-rule), in which from a given permutation any two jobs can be interchanged to get another permutation. The following example shows that for a given f CBS-rule may not produce a convex graph, while BS-rule may.

Example 1.3.1

Consider $f(j_1, j_2, j_3) = c_{j_1 j_2} + c_{j_2 j_3}$ for any typical permutation (j_1, j_2, j_3) of (1,2,3) where c_{ij}'s are given by the table below.

c_{ij}

i \ j	1	2	3
1	-	2	4
2	2	-	3
3	3	2	-

The corresponding preference graph for the example using CBS-rule is given in Fig. 1.3.1 with f(x) above the nodes, and node number in the boxes along with the corresponding permutation.

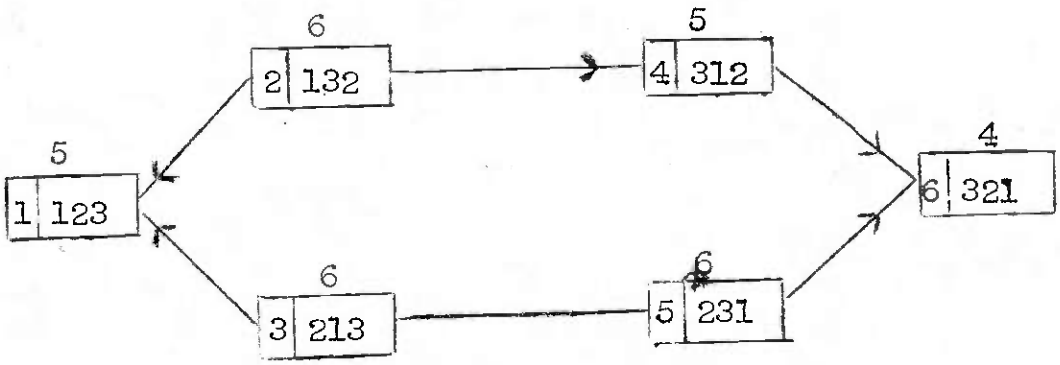


Figure 1.3.1

It is easy to see that this graph is not convex. Now if we use BS-rule, nodes (1) & (6) are adjacent and we introduce the corresponding arc. Similarly nodes (2) & (5) and (3) & (4) are adjacent. Thus we get the preference graph given in Fig. 1.3.2. Now, we observe that the three conditions given in the definition 1.3.3 are satisfied for $B = \{1,2,3,4,5\}$ and $R = \{6\}$. Thus the preference graph is convex.

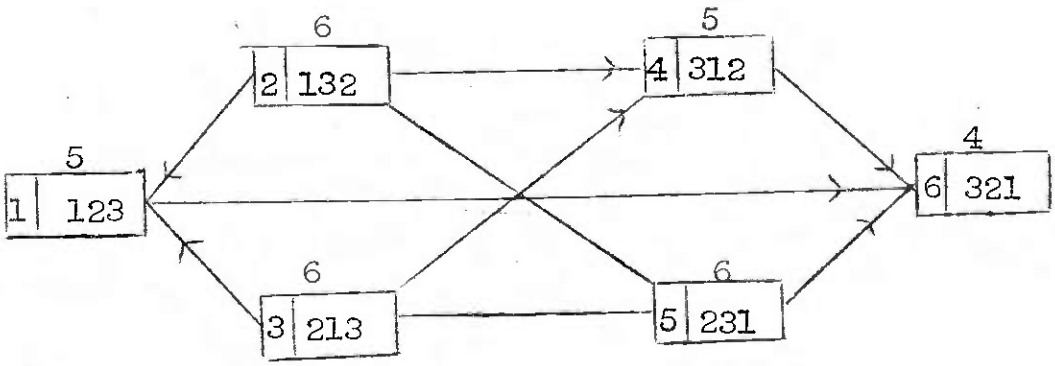


Figure 1.3.2

It is interesting to note that parameters of the problem, like processing times, set up costs do play an essential role in obtaining simple switching rules for solving the problem. Johnson's special cases of the three stage problem are examples in which, for the $(n/3/F/F_{max})$ problem in general CBS-rule need not produce a convex graph, whereas if $\min_i A_i \geq \max_i B_i$ or $\min_i C_i \geq \max_i B_i$, CBS-rule yields a convex graph. John G. Rau's (1971) result given in the next sub-section is another interesting case, where CBS-rule works.

Elmaghraby, in his paper (1971), makes the following assertion : If the nodes of a directed graph are partitioned into two sets B and R such that

(i) if B is not empty then every node in B has at least one arrow out of it in the direction of improvement of f, i.e. $\forall\, x \in B$, $\Gamma^{1}(x) \neq \phi$, then the graph is convex.

However the following example shows that this assertion is wrong.

Example 1.3.2 Consider all permutations with (1,2,3) with

$f(j_1, j_2, j_3) = c_{j_1 j_2} + c_{j_2 j_3}$ for any permutation (j_1, j_2, j_3)

where c_{ij}'s are as given in the following table.

c_{ij}

i \ j	1	2	3
1	-	4	5
2	6	-	5
3	1	4	-

The preference graph for this example is given in Fig. 1.3.3.

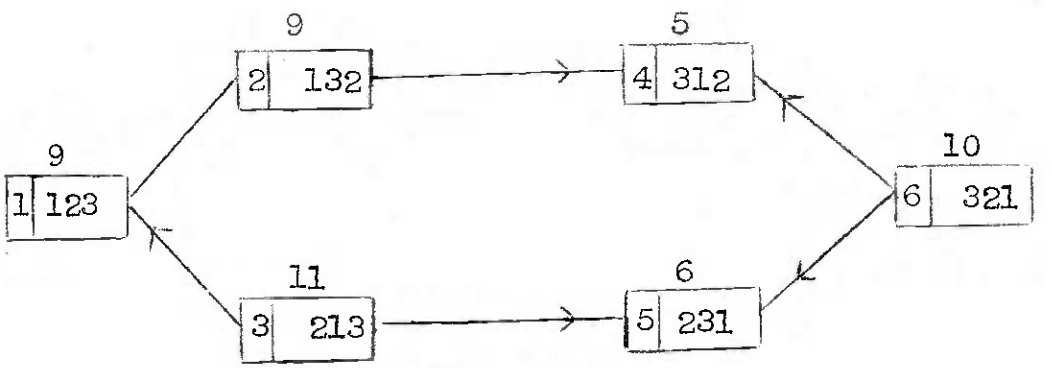


Figure 1.3.3

With $B = \{2,3,6\}$ and $R = \{1,4,5\}$, it is easily seen that condition (i) above is satisfied whereas G is not convex, since

$$f(1) = 9 \neq f(4) = 5 \neq f(5) = 6$$

Similarly Assertion 3 in Elmaghraby (1971) is also not correct as $W = \{2,3,6\}$; but $f(3) = 11 \neq f(6) = 10 \neq f(2) = 9$ where W is the collection of nodes such that $W \subseteq B$ and $x \in W \Rightarrow$ There does not exist $y \in Q$ such that (y,x) is a directed arc.

1.3.2 A Note on John G. Rau's Result (1971)

Rau (1971) considers the minimization of the function $f(\sigma) = \sum_{s=1}^{n} h(j_s)\, g_s(j_1 \ldots j_s)$ over all permutations, $\sigma = (j_1 \ldots j_m)$ of $(1,2,\ldots,n)$ where $h : \{1,2,\ldots,n\} \to R$ and

$g_s : \underline{P}_s \to R$ with $\underline{P}_s$ denoting the set of all permutations of s distinct elements from $\{1,2,\ldots,n\}$.

The weighted tardiness problem in a single machine shop i.e. $(n/1/F/T_u)$ has an objective function like f given above, but it is known that in general even BS-rule doesn't lead to an optimal solution in this case. Under certain conditions Rau shows that CBS-rule leads to an optimal solution of the problem stated above. We consider his result and show the equivalence of his conditions to some simpler ones.

Definition 1.3.4 : We say functions g_s are Markovian (or satisfies condition A) in case,

$$g_s(x_1 x_2 \ldots x_s) = g_s(y_1 y_2 \ldots y_{s-1}, x_s) \text{ for all } 1 \leq s \leq n$$

and for all permutations $(y_1 y_2 \ldots y_{s-1})$ of $(x_1 x_2 \ldots x_{s-1})$.

Definition 1.3.5 : We say functions g_s satisfy condition B, in case, there exists a real valued function $G : \{1,2 \ldots n\} \to R$

and positive functions $G_1, G_2 \ldots G_n$ with $G_1 = 1$ and $G_s : \underline{P}_{s-1} \to R$ such that for $1 \leq s \leq n-1$, for all sets of distinct integers $\{x_1, x_2 \ldots x_{s-1}\}$ and distinct integers $\{y_1, y_2 \ldots y_s\}$ which are subsets of $\{1, 2 \ldots n\}$ with the property that

$$\{x_1\ x_2 \ldots x_{s-1}\} = \{y_1\ y_2 \ldots y_s\} - \{y_j\},$$

we have

$$g_s(x_1 \ldots x_{s-1}\ x_s) - g_{s+1}(y_1\ y_2 \ldots y_s,\ x_s) = G(y_j)\ G_s(x_1\ x_2 \ldots x_{s-1}).$$

Theorem 1.3.1 : If h is a positive function and $g_s, 1 \leq s \leq n$ are functions such that they satisfy condition A and condition B then the minimum of

$$f(\sigma) = \sum_{s=1}^{n} h(j_s)\ g_s(j_1 \ldots j_s)$$

is attained at $(j_1^*, j_2^* \ldots j_n^*)$ such that

$$G(j_1^*)/h(j_1^*) \geq \ldots \geq G(j_n^*)/h(j_n^*).$$

Before we proceed to give a proof of this theorem, we define conditions C and D and show the relationships between A, B, C and D.

Definition 1.3.6 : We say functions g_s satisfy condition C, in case, there exist a real valued function $G : \{1,2 \ldots n\} \to R$ and positive functions $G_1 \ldots G_n$ with $G_1 = 1$ and $G_s : \underline{P}_{s-1} \to R$ such that for $1 \leq s \leq n$ and for all $(y_1 \ldots y_s) \in \underline{P}_s$ and $x_s \neq y_j \ \forall \ 1 \leq j \leq s$, we have

$$g_s(y_1 \ldots y_{s-1}, x_s) - g_{s+1}(y_1 \ldots y_s, x_s) = G(y_s)\, G_s(y_1 \ldots y_{s-1})$$

Definition 1.3.7 : We say functions g_s satisfy condition D, in case,

$$g_s(x_1 \ldots x_s) = g_s(y_1 \ldots y_{s-1}, x_s) \text{ for all } 1 \leq s \leq n$$

and for all contiguous binary permutations $(y_1 \ldots y_{s-1})$ of $(x_1 \ldots x_{s-1})$.

Lemma 1.3.1 : We have,

(I) Condition D $\Longleftrightarrow$ Condition A

(II) Condition B $\Rightarrow$ Condition C

(III) Condition B $\Rightarrow$ Condition A

(IV) Condition C $\not\Rightarrow$ Condition B

(V) Condition C and D together $\Rightarrow$ Condition B.

Proof : (I) A $\Rightarrow$ D is trivial. To prove D $\Rightarrow$ A note that any permutation of $(x_1 \ldots x_{s-1})$ can be obtained from $(x_1 \ldots x_{s-1})$ by a series of CB-permutations.

(II) B $\Rightarrow$ C is immediate from the definitions of Conditions B and C.

(III) B $\Rightarrow$ A.

If g_s's satisfy Condition B, then

$$g_s(x_1 \dots x_{s-1}, x_s) - g_{s+1}(y_1 \dots y_s, x_s) = G(y_j).\ G_s(x_1 \dots x_{s-1})$$

where $\{y_j\} = \{y_1 \dots y_s\} - \{x_1 \dots x_{s-1}\}$

i.e. $$g_{s+1}(y_1 \dots y_s, x_s) = g_s(x_1 \dots x_{s-1}, x_s) - G(y_j).\ G_s(x_1 \dots x_{s-1})$$

Now to show that the functions g_s satisfy Condition A, we have to show

$$g_{s+1}(y_1 \dots y_s, x_s) = g_{s+1}(z_1 \dots z_s, x_s)$$

where $(z_1 \dots z_s)$ is a permutation of $(y_1 \dots y_s)$, for $1 \le s \le n-1$.

This is so, because, if

$$\{x_1 \dots x_{s-1}\} = \{y_1 \dots y_s\} - \{y_j\} = \{z_1 \dots z_s\} - \{y_j\}$$

then

$$g_{s+1}(y_1 \dots y_s, x_s) = g_s(x_1 \dots x_{s-1}, x_s) - G(y_j)\ G_s(x_1 \dots x_{s-1})$$

and

$$g_{s+1}(z_1 \dots z_s, x_s) = g_s(x_1 \dots x_{s-1}, x_s) - G(y_j)\, G_s(x_1 \dots x_{s-1})$$

Thus Condition A is satisfied for g_s, $2 \leq s \leq n$.

For g_1 Condition A is trivially satisfied.

(IV) C $\not\Rightarrow$ B. To show this we consider the following example : Let $n = 3$

$g_1(1) = 4$ $\quad g_2(1\ 2) = 5$ $\quad g_2(2\ 1) = 7$

$g_1(2) = 3$ $\quad g_2(1\ 3) = 7$ $\quad g_2(3\ 1) = 7$

$g_1(3) = 5$ $\quad g_2(2\ 3) = 8$ $\quad g_2(3\ 2) = 6$

$g_3(123) = 22$ $\quad g_3(231) = 16$

$g_3(132) = 20$ $\quad g_3(321) = 19$

$g_3(213) = 14$ $\quad g_3(312) = 14$

For these g_s's we can find G, and G_i's as given below satisfying Condition C

$G(1) = -2$, $\quad G(2) = -3 = G(3)$

$G_1 = 1$; $\quad G_2(1) = 5$,

$G_2(2) = 3$,

and $\quad G_2(3) = 4$.

With $x_s = 3$ and $\{y_1, y_2\} = \{1,2\}$ we require, if Condition B were to be true, $g_3(123) = g_3(213)$. But we have,

$$g_3(123) = 22 \neq g_3(213) = 14$$

Hence C $\not\Rightarrow$ B.

(V) If g_s's satisfy both Conditions C and D then B is satisfied.

Now,

$$g_{s+1}(y_1 \ldots y_s, x_s) = g_{s+1}(u_1 \ldots u_s, x_s) \qquad \ldots.(1)$$

for any permutation $(u_1 \ldots u_s)$ of $(y_1 \ldots y_s)$ since Condition D $\Rightarrow$ Condition A.

Let $z_1 \ldots z_{s-1}$ be any permutation of $y_1 \ldots y_{s-1}$

then $g_{s+1}(y_1 \ldots y_s, x_s) = g_s(y_1 \ldots y_{s-1}\, x_s)$

$$- G(y_s)\, G_s(y_1 \ldots y_{s-1})$$

from Condition C

and $g_{s+1}(z_1 \ldots z_{s-1}, y_s, x_s) = g_s(z_1 \ldots z_{s-1}, x_s)$

$$- G(y_s)\, G_s(z_1 \ldots z_{s-1})$$

from Condition C

Using (1) we have

$$G_s(y_1 \ldots y_{s-1}) = G_s(z_1 \ldots z_{s-1}).$$

Now consider

$$g_s(x_1 \cdots x_{s-1}, x_s) - G(y_j)\, G_s(x_1 \cdots x_{s-1})$$

$$= g_s(y_1 \cdots y_{s-1}, x_s) - G(y_j)\, G_s(y_1 \cdots y_{s-1})$$

$$= g_{s+1}(y_1 \cdots y_{s-1}\, y_j, x_s)$$

$$= g_{s+1}(y_1 \cdots y_j \cdots y_s, x_s) \quad \text{from Condition A.}$$

This proves Condition B is satisfied.

This completes the proof of the lemma.

Thus it is sufficient to verify whether Condition B holds, to apply Theorem 1.3.1. Further, C and D can be verified, instead of verifying B, as it requires less amount of computation. We now give a proof of the theorem 1.3.1, where instead of Conditions A and B we assume Conditions C and D. We note that the proof is similar to Rau's (1971). Let h and g_s be as defined earlier.

Proof : Let $Q = (j_1 \cdots j_n)$

$$f(Q) = \sum_{\substack{s=1 \\ s \neq i, i+1}}^{n} h(j_s)\, g_s(j_1 \cdots j_s)$$

$$+ h(j_i)\, g_i(j_1 \cdots j_i)$$

$$+ h(j_{i+1})\, g_{i+1}(j_1 \cdots j_i\, j_{i+1})$$

Interchanging j_i and j_{i+1} we have $Q' = (j_1 \cdots j_{i+1}\ j_i \cdots j_n)$

Now

$$f(Q') = \sum_{\substack{s=1 \\ s\neq i, i+1}}^{n} h(j_s)\, g_s(j_1 \cdots j_s)$$

$$+ h(j_{i+1})\, g_i(j_1 j_2 \cdots j_{i-1}\ j_{i+1})$$

$$+ h(j_i)\, g_{i+1}(j_1 j_2 \cdots j_{i-1}\ j_{i+1}\ j_i).$$

Let $g'(j_i, j_{i+1}) = h(j_i)\, g_i(j_1\ j_2 \cdots j_i)$

$$+ h(j_{i+1})\, g_{i+1}(j_1\ j_2 \cdots j_i\ j_{i+1})$$

and $g'(j_{i+1}, j_i) = h(j_{i+1})\, g_i(j_1 j_2 \cdots j_{i-1}\ j_{i+1})$

$$+ h(j_i)\, g_{i+1}(j_1 j_2 \cdots j_{i-1}\ j_{i+1}\ j_i)$$

Therefore

$$f(Q) - f(Q') = g'(j_i, j_{i+1}) - g'(j_{i+1}, j_i)$$

because of Condition D

$$= h(j_i)\, G(j_{i+1})\, G_i(j_1 \cdots j_{i-1})$$

$$- h(j_{i+1})\, G(j_i)\, G_i(j_1 \cdots j_{i-1})$$

because of Condition C

$$= [h(j_i)\, G(j_{i+1}) - h(j_{i+1}) G(j_i)]\, G_i(j_1 j_2 \cdots j_{i-1})$$

But $G_i(j_1 \cdots j_{i-1}) > 0$ by Condition C

Therefore

$$f(Q) - f(Q') \leq 0 \quad \text{iff} \quad g'(j_i, j_{i+1}) \leq g'(j_{i+1}, j_i)$$

$$\text{i.e.} \quad \text{iff} \quad h(j_i)\, G(j_{i+1}) \leq h(j_{i+1})\, G(j_i)$$

$$\text{i.e.} \quad G(j_i)/h(j_i) \geq G(j_{i+1})/h(j_{i+1}).$$

Hence the theorem 1.3.1 is proved.

Corollary 1.3.1 : If h is any real valued function and g's satisfy Conditions C and D then

$$f(Q) - f(Q') \leq 0 \quad \text{iff} \quad h(j_i)\, G(j_{i+1}) \leq h(j_{i+1})\, G(j_i).$$

Thus f as defined here, satisfies the sufficient condition of Smith's theorem (1956) and $g(i,j) = h(i)\, G(j)$; hence CBS-rule will produce an optimal sequence. Of course, this $g(\cdot,\cdot)$ is not transitive, i.e.

$$g(i,j) \leq g(j,i) \text{ and } g(j,k) \leq g(k,j) \not\Rightarrow g(i,k) \leq g(k,i),$$

as shown by the example below :

Example 1.3.3 : $h(1) = 3 \quad G(1) = -5$

$h(2) = -4 \quad G(2) = 1$

$h(3) = 2 \quad G(3) = 6$

$$g(1,2) = h(1)\ G(2) = 3 \times 1 = 3$$

$$g(2,1) = h(2)\ G(1) = -4 \times -5 = 20$$

$$g(1,2) \leq g(2,1) \quad \text{i.e.} \quad 1 \leftarrow 2$$

Similarly $g(2,3) = -24 \leq 2 = g(3,2)$ i.e. $2 \leftarrow 3$.

But $g(1,3) = 3 \times 6 > g(3,1) = -5 \times 2$ i.e. $1 \not\leftarrow 3$.

Hence transitivity does not hold.

1.3.3 A 'good' Algorithm for Non-transitive Case

In this section we give a simple 'good' algorithm to find an optimal sequence for the case when f satisfies the sufficient condition of Smith's theorem, but the corresponding function $g(\cdot,\cdot)$ is non-transitive. It is often mentioned that if CBS-rule can produce an optimal sequence then starting with any sequence we can reach an optimal sequence by repeatedly using $g(\cdot,\cdot)$'s and the inequalities in them. But this procedure may take a maximum of as many as n! steps. Hence an algorithm which is 'good', in the sense that the maximum number of steps required is only a polynomial in n [See for the definition of a 'good' algorithm in Jack Edmonds (1965)], is attempted in what follows. This takes at most $\frac{n(n-1)}{2} - 1$ steps, where a step is a comparison and calculation

of $g(i,j)$ and $g(j,i)$ for some i and j. Hence it is a 'good' algorithm for the non-transitive case. In case of transitive $g(\cdot,\cdot)$ there are better algorithms, see Johnson (1954), Mc Naughton (1959).

The Algorithm

Step 1 : Set $r = 2$. Let $\bar{p}$ be the sequence of the first r jobs such that $g(j_i, j_{i+1}) \leq g(j_{i+1}, j_i)$ $\forall$ $i=1,2...r-1$. Go to Step 2.

Step 2 : Set $i = 1$.
If $g(r+1, j_i) \leq g(j_i, r+1)$ go to Step 3. Otherwise go to Step 4.

Step 3 : Set $\bar{p} = (r+1, \bar{p})$. Go to Step 3.a.

Step 3.a : If $(r+1) < n$ go to Step 2. Otherwise go to Step 7.

Step 4 : If $i < r$, go to Step 4.a. Otherwise go to Step 5.

Step 4.a : Set $i = i+1$. If $g(r+1, j_i) \leq g(r+1, j_i)$ go to Step 6. Otherwise go to Step 4.

Step 5 : Set $\bar{p} = (\bar{p}, r+1)$. Go to Step 3.a.

Step 6 : Set $\bar{p} = (j_1 \ldots j_{i-1}\ r+1\ j_i \ldots j_r)$, go to Step 3.a.

Step 7 : $\bar{p}$ is optimal. Stop.

C H A P T E R 2

FLOW SHOP PROBLEMS - II (GENERAL OBJECTIVES)

2.0 Introduction

Total elaspsed time minimization in the Flow-shop context was the topic of discussion in the last chapter. Minimization problems with general objective functions are considered in this chapter. In section 2.1 we generalize the dominance rule known for the total elapsed time objective (Ignall and Schrage (1965), Szwarc (1971)) to more general objectives. In section 2.2 we discuss the (n/m/F/u(T)) problem, in which minimization of certain non-decreasing functions of job tardiness is required. This problem has been considered by Gupta (1971), Elmaghraby (1968) and Hamilton Emmons (1969), to mention a few. The algorithm given by Brown and Lomnicki (1966) is slightly modified to solve the problem. In section 2.3 some special cases of ($n/2/F/C_u$) problem are considered. Section 2.4 deals with some special cases of (n/m/F/f) problem with f, a regular measure of performance. In 1968 Elmaghraby considered the ($n/1/F/T_u$) problem. This problem is discussed in section 2.5. We also extend some results of Hamilton Emmons (1969). Section 2.6 deals with the minimization of weighted tardiness for single machine sequencing problem with common dead line. Finally, in

section 2.7 minimizing number of late jobs with intermittent job arrivals is solved, for a single stage shop with common dead line.

2.1 A General Dominance Theorem

Dominance rule for the minimization of the total elapsed time is well known (Szwarc (1971), Ignall and Schrage (1965)). Here we generalize that to more general objective functions that have the property A defined below.

Now, let us consider the (n/m/F/f) problem with the no passing restriction where the objective function is $f : R^n \to R$. Let $C_1, C_2, \ldots, C_n$ be the completion times of jobs $1, 2, \ldots n$ respectively for a typical sequence. Let $C = (C_1, C_2, \ldots, C_n)$. Let $C_\lambda = (C_{i_1}, C_{i_2}, \ldots, C_{i_r})$ for any $\lambda = (i_1, i_2, \ldots, i_r)$, an ordered subset of $N = \{1, 2, \ldots, n\}$. Let λ and μ be any two disjoint, ordered subsets of N such that $\lambda(\_)\mu = N$, $\lambda \neq \phi$, $\mu \neq \phi$.

Definition 2.1.1 : We say f has property A in case

(i) There exist real valued functions g_λ, h_λ for all λ and a binary operation o on the real line which preserves inequalities such that

$$f(C) = g_\lambda(C_\lambda) \text{ o } h_\mu(C_\mu) \quad \forall \ \lambda \text{ and } \mu$$

(ii) $h_\mu(.)$ is a real valued function such that for x, x' with $x_i \leq x'_i \ \forall\ i$, $h_\mu(x) \leq h_\mu(x') \ \forall\ \mu$.

As in chapter 1 let $t(\lambda, k)$ denote the earliest time at which jobs in λ can be completed on all the machines upto k.

Theorem 2.1.1 : Let $\sigma = (j_1 \ldots j_r)$ and $\sigma' = (s_1 \ldots s_r)$ be two permutations of the same subset of r jobs. Let π be any sequence $(j_{r+1}, \ldots, j_n)$ of the remaining $n-r$ jobs. Let the vector of completion times corresponding to $\sigma\pi$ be $C = (C_1 \ldots C_n)$ and let $C' = (C'_1, C'_2 \ldots C'_n)$ be that of $\sigma'\pi$. Let f have property A. If $g_\sigma(C_\sigma) \leq g_{\sigma'}(C'_{\sigma'})$ and

$$t(\sigma, k) \leq t(\sigma', k) \quad \text{for all } k,\ 1 \leq k \leq m$$

then

$$f(C) \leq f(C') \quad \text{and},$$

$$t(\sigma\pi', k) \leq t(\sigma'\pi', k) \quad \text{for all } k,\ 1 \leq k \leq m$$

and for any ordered subset π' of $N - \sigma$.

Proof : First we note that

$$t(\sigma, k) \leq t(\sigma', k) \quad \text{for all } k,\ 1 \leq k \leq m$$

$$\Rightarrow\ t(\sigma\pi', k) \leq t(\sigma'\pi', k) \quad \text{for all } k,\ 1 \leq k \leq m$$

$$\text{and for any } \pi' \subseteq N-\sigma \qquad \ldots\ 2.1.1$$

This, actually is the dominance theorem for the total elapsed time. For a proof of this see (Szwarc (1971)).

Now consider,

$$f(C) = f(C_1, C_2, \ldots C_n)$$

$$= g_\sigma(C_\sigma) \circ h_\pi(C_\pi)$$

$$\leq g_{\sigma'}(C'_{\sigma'}) \circ h_\pi(C_\pi)$$

by property A and the given fact

$$g_\sigma(C_\sigma) \leq g_{\sigma'}(C'_{\sigma'})$$

But from 2.1.1 we have,

$$C_{j_{r+q}} = t(\sigma\ (j_{r+1}\ j_{r+2} \cdots j_{r+q}), m) \leq t(\sigma'\ (j_{r+1}\ j_{r+2} \cdots j_{r+q}), m)$$

$$= C'_{j_{r+q}}$$

for all q such that $1 \leq q \leq n-r$

i.e. $C_\pi \leq C'_\pi$.

Therefore,

$$h_\pi(C_\pi) \leq h_\pi(C'_\pi)$$

(by property A)

And so,

$$f(C) \leq g_{\sigma'}(C'_{\sigma'}) \circ h_\pi(C_\pi)$$

$$\leq g_{\sigma'}(C'_{\sigma'}) \circ h_\pi(C'_\pi)$$

$$= f(C'_1 \ldots C'_n) = f(C')$$

This completes the proof of the theorem.

The property A is satisfied by many of the objectives considered in the literature.

Example : $f = F_{max}$. Here

(i) $f(C_1 \ldots C_n) = \text{Max}(C_1 \ldots C_n)$

(ii) $a \circ b = \text{Max}(a, b)$

and (iii) $g_\sigma(C_j) = \underset{j\varepsilon\sigma}{\text{Max}} \; C_j$

and $h_\pi(C_\pi) = \underset{j\varepsilon\pi}{\text{Max}} \; C_j$

Hence, f has the property A.

Similarly, we see that total tardiness, average tardiness, cost of tardiness as discussed in section 2.2, number of late jobs and any separable function that is nondecreasing in each variable, have the property A.

The above theorem is useful in discarding certain partial sequences from further consideration while searching for a complete sequence. Thus this dominance rule can be incorporated in any Branch and Bound algorithm in which partial sequences form the nodes of the tree corresponding to the problem.

2.2 Minimization of Certain Functions of Job Tardiness

2.2.0 Introduction :

We consider the problem $(n/m/F/u(T))$ where $u(T)$ is a nondecreasing function of T given by

$$u(T) = \sum_{j=1}^{n} u_j(T_j), \quad \text{where} \quad T_j \text{ is the}$$

tardiness for job j. i.e., $T_j = \text{Max}(C_j - d_j, 0)$, C_j being the completion time of job j in a sequence and d_j, the dead line for job j. The usual assumptions A.1 to A.8 are made. This problem has been considered by Gupta (1971), Nabashima (1971) and Elmaghraby (1968). Gupta (1971) considers such a problem and develops a partial search procedure in which no bounds are used for selecting partial sequences, but a partial sequence is extended as long as the cost of it is less than the cost of a 'so far best' complete sequence. Nabashima (1971) gives a general algorithm which solves problems with such objective functions. He illustrates the algorithm, with a numerical example, for the problem of minimizing mean tardiness. Elmaghraby (1968) considers $(n/1/F/T_u)$ problem with different dead lines and gives a Branch and Bound algorithm.

In section 2.2.1 we find simple bounds for cost of tardiness for any complete sequence obtained from a partial sequence, and use them in the algorithm developed for solving the (n/m/F/u(t)) problem. This algorithm is similar to Brown and Lomnicki's (1966) for minimizing F_{max}. Certain special cases when $u(T)$ satisfies some further conditions are discussed in section 2.2.2.

2.2.1 Main Results, Lower Bounds and an Algorithm

A sequence σ of q jobs is called a partial sequence if $q < n$. When $q = n$ it is called a complete sequence.

Let σ' denote any complete sequence obtained from a partial σ i.e., $\sigma' = (\sigma\pi)$ where π is a sequence of all jobs in $N - \sigma$.

Let $p^{(1)} \leq \ldots \leq p^{(n-q)}$ be the ordered processing times on the last machine for jobs in π.

Lemma 2.2.1 Given $\pi = (j_1 \ldots j_{n-q})$ any sequence of jobs in $N - \sigma$, the completion times satisfy

$$C_{j_v} \geq t(\sigma,m) + \sum_{r=1}^{v} p^{(r)} \quad \text{for} \quad 1 \leq v \leq n-q$$

when $\sigma' = (\sigma\pi)$ is the sequence considered.

Moreover,

$$C_{j_{n-q}} = t(\sigma', m) \geq L(\sigma)$$

where $L(\sigma)$ is a lower bound for total elapsed time for any completion of σ.

The proof of this lemma is easy and so is omitted.

A bound $L(\sigma)$ can be obtained using the following expression given in Brown and Lomnicki (1966).

$$L(\sigma) = \underset{1 \leq i \leq m}{\text{Max}} \; L_i(\sigma) \qquad \ldots \quad 2.2.1$$

where,

$$L_i(\sigma) = t(\sigma, i) + \sum_{j \in N-\sigma} p_{ij} + \underset{j \in N-\sigma}{\text{Min}} \left[\sum_{r=i+1}^{m} p_{rj} \right]$$

Henceforth we use $L(\sigma)$ as defined above.

Lemma 2.2.2 For any completion σ' of the partial sequence σ, we have

$$C(\sigma') \geq \bar{C}(\sigma) = C(\sigma) + \sum_{\substack{j \in N-\sigma \\ j \neq j_o}} u_j(T'_j) + u_{j_o}(T'_{j_o}{}')$$

where $C(\sigma) = \sum_{j \in \sigma} u_j(T_j)$ and $T'_j = \text{Max}[t(\sigma, m) + p_{mj} - d_{j,o})$

and $T'_j{}' = \text{Max}[L(\sigma) - d_j, 0]$ and j_o is such that,

$$u_{j_o}(T'_{j_o}) - u_{j_o}(T^!_{j_o}) = \underset{j\in N-\sigma}{\text{Min}} \left[u_j(T'_j) - u_j(T^!_j)\right]$$

Proof : We have

$$C_j \geq t(\sigma,m) + p_{mj}, \quad j \in N-\sigma.$$

And u_j's are nondecreasing since $u(T)$ is nondecreasing. Therefore,

$$\sum_{j\in N-\sigma} u_j(T_j) \geq \sum_{j\in N-\sigma} u_j(T^!_j)$$

Since $C_{j_{n-q}} = t(\sigma\pi, m) \geq L(\sigma)$ for any completion of σ, we have

$$T_{j_{n-q}} \geq T'_{j_{n-q}} = \text{Max}(L(\sigma) - d_{j_{n-q}}, 0)$$

$$\geq T^!_{j_{n-q}} \quad (\text{Since } t(\sigma,m) + p_{mj_{n-q}} \leq L_m(\sigma))$$

Thus by the choice of j_o as given in the lemma, the result follows.

The above lemma gives a lower bound for the total cost of tardiness for any σ'. This lower bound is used in an algorithm given below for solving the (n/m/F/u(T)) problem.

An Algorithm :

The set of all partial sequences can be represented as the nodes of a tree with root at the partial sequence (.)

corresponding to $q = 0$. A pair of nodes σ and σ_1 are connected by an edge in case, $\sigma_1 = (\sigma j)$ for some $j \in N-\sigma$. There are $n-q$ descendants for a partial sequence σ with q jobs. The lower bound associated with any node is non-decreasing as we move away from the root and also gives the minimum possible actual cost for a terminal node corresponding to a complete sequence.

This algorithm is same as that given in Brown and Lomnicki (1966), except for certain additional computations made while calculating the bounds.

Step 1 : Start with node (.) with $q = 0$.

Let $\sigma=(\cdot)$. Go to Step 2.

Step 2 : Generate the $n - q$ descendants of σ by fixing j in the $q+1^{st}$ position, one by one systematically for $j \in N-\sigma$. Let τ_i for $1 \leq i \leq n-q$ be the nodes thus generated. Go to Step 3.

Step 3 : For each τ_i workout the lower bound $\bar{C}(\tau_i)$ given by Lemma 2.2.2 calculating $L(\tau_i)$ given by 2.2.1. Store $(\tau_i, \bar{C}(\tau_i), q+1)$ for each τ_i. Go to Step 4.

Step 4 : Choose a node from among all nodes available for branching with the smallest value of $\bar{C}$. Let σ be the node so chosen with q jobs. If $q < n-1$ go to Step 2. Otherwise go to Step 5.

Step 5 : $\sigma^* = (\sigma j)$ is optimal with cost $\bar{C}(\sigma)$, where j is the only job not in σ. Stop.

We illustrate the use of this algorithm by solving the following problem.

Example 2.2.1 The following table gives the necessary data on processing times and dead lines. The cost functions are given below.

Job \ Machine	Processing time					Total	Dead line
	1	2	3	4	5		
1	4	3	7	2	8	24	28
2	3	7	2	8	5	25	33
3	1	2	4	3	7	17	17
4	3	4	3	7	2	19	20
Total	11	16	16	20	22		

$$u_1(x) = 2x^2$$
$$u_2(x) = 3x$$
$$u_3(x) = 4x$$
$$u_4(x) = x^2+2x, \quad x \geq 0$$

We are interested in minimising total cost of tardiness. The functions given are obviously nondecreasing functions. With the node $\sigma = (.)$ we enter Step 2 and generate the four nodes corresponding to $\tau_1 = (1)$, $\tau_2 = (2)$, $\tau_3 = (3)$ and $\tau_4 = (4)$. The table below gives the details of the calculations required to find $L(\tau_i)$ as given by 2.2.1.

$t(\sigma,k)$	0	0	0	0	0						
k	1	2	3	4	5	1	2	3	4	5	$L(\tau_i)$
τ_i	$t(\tau_i, k)$					L_k					
(1)	4	7	14	16	24	27	32	32	36	38	38
(2)	3	10	12	20	25	27	31	35	34	42	42
(3)	1	3	7	10	17	27	29	28	29	32	32
(4)	3	7	10	17	19	27	33	33	35	39	39

We calculate T_j^i, $T_j^{i'}$ for $j \in N-\tau_i$ for different τ_i. Consider $\tau_1 = (1)$. We have $t(\tau_1, m) = 24$ and $L(\tau_1) = 38$ from the above table. The following table gives T_j^1 and $T_j^{1'}$ for τ_1.

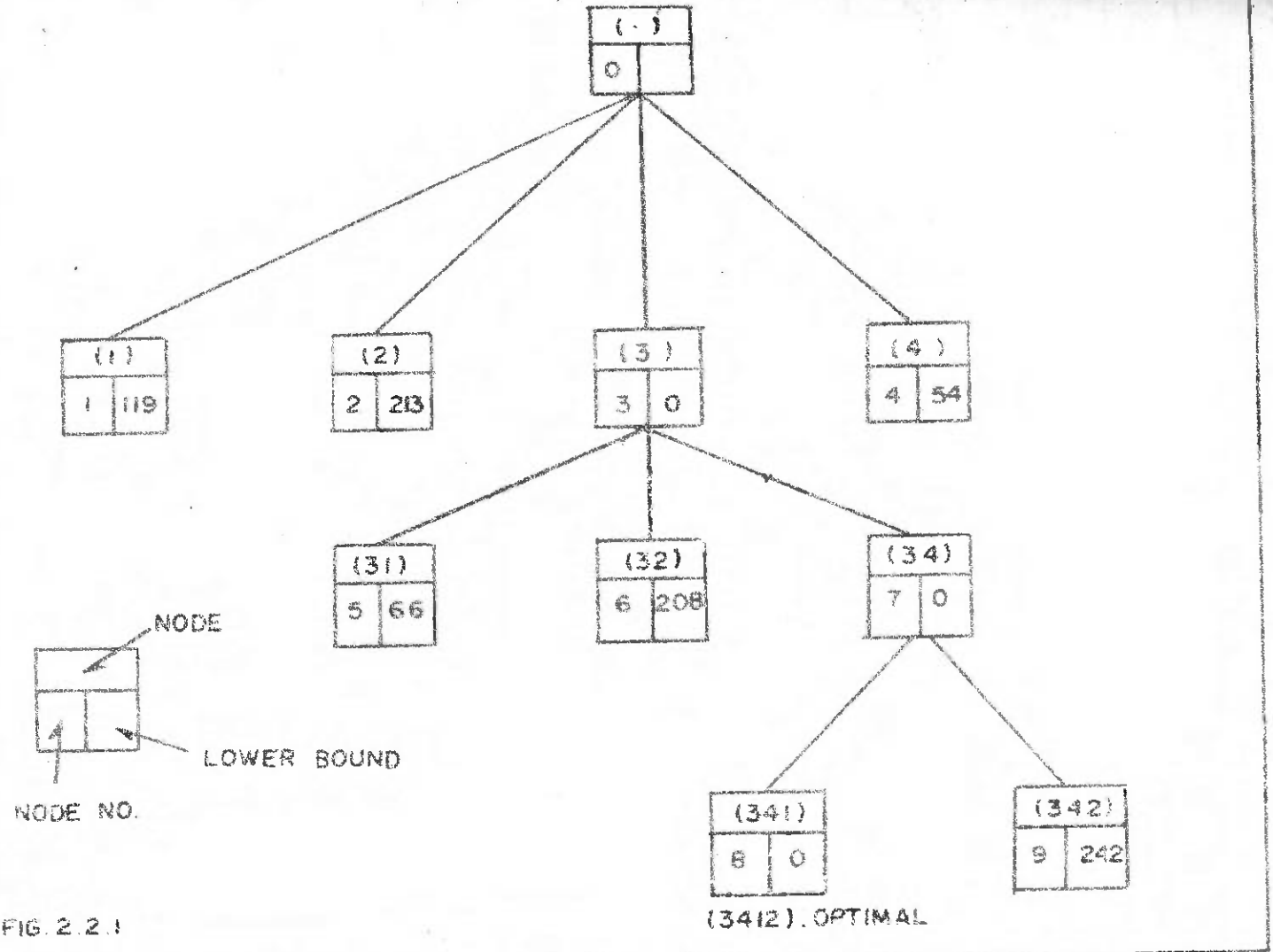

(.)
0
(1)
1
119
(2)
2
213
(3)
3
0
(4)
4
54
(31)
5
66
(32)
6
208
(34)
7
0
(341)
8
0
(342)
9
242
NODE
LOWER BOUND
NODE NO.
(3412). OPTIMAL

FIG. 2.2.1

j	$T_j^!$	$T_j^{!'}$	$U_j(T_j^!)$	$U_j(T_j^{!'})$
2	0	5	0	15
3	14	21	56	84
4	6	18	48	360
		Total :	104	

We have $\bar{C}(\tau_1) = 0 + 104 + 15 = 119$.

Similarly we find $\bar{C}(\tau_2) = 213$, $\bar{C}(\tau_3) = 0$ and $\bar{C}(\tau_4) = 54$. So we branch, taking $\sigma = (3)$. We find the nodes created are (31), (32), (34) with $\bar{C}(31) = 66$, $\bar{C}(32) = 208$ and $\bar{C}(34) = 0$. And so we branch from $\sigma = (34)$. Nodes (342) and (341) are generated with $\bar{C}(342) = 242$ and $\bar{C}(341) = 0$. As $q = n-1$ we stop here. An optimal sequence is (3412) with cost 0. The tree generated for the problem is given in Fig. 2.2.1.

Remarks

1. At any stage while choosing a node σ for branching, if $\bar{C}(\sigma) << \bar{C}$ for all other available nodes for branching, then while calculating $\bar{C}(\tau_i)$, instead of $L(\sigma)$ we can use $L_m(\sigma)$. This reduces the computation considerably. In the above example $\bar{C}(31) = 66$, but we get using $L_m(\sigma)$ the modified

lower bound for (31) to be

$$LB(31) = 0 + 63 + 0 = 63.$$

2. $L_m(\sigma)$ coincides with $L(\sigma)$ and is equal to $\sum_{j=1}^{n} p_j$ for the single machine case.

3. The dominance theorem proved in section 2.1 is applicable in this case. The theorem when applied to this objective function goes as follows :

The sequences starting with σ' can be discarded while searching for an optimal sequence in case $C(\sigma) \leq C(\sigma')$ and $t(\sigma,k) \leq t(\sigma',k)$ for all k such that $1 \leq k \leq m$ where σ and σ' are two different permutations of the same subset of jobs.

2.2.2 Some Special Cases of the Objective Function u(T)

In this section we consider u_j's satisfying in addition to nondecreasing property, the following conditions (after renaming the jobs if necessary).

(i) $u_j(x) \leq u_{j+1}(x)$ for $j = 1,2 \ldots n-1$, $x \in [0, \beta]$

where β is sufficiently large.

and (ii) Given $y \geq x \geq 0$ and $\omega \geq 0$

$$u_{j+1}(y + \omega) - u_{j+1}(y) \geq u_j(x + \omega) - u_j(x)$$

for $j = 1,2 \ldots, n-1$. 2.2.2

These two assumptions only mean the penalty functions behave 'like' linear penalty functions, for which these are satisfied. In cases where $u(T)$ has the above properties we can improve the lower bound for the total penalty cost for any complete sequence obtained from a partial sequence. To achieve this, we need the following lemma 2.2.3 which gives an optimal solution for a special type of assignment problems.

Definition 2.2.1 : We say matrix C has non-decreasing rows in case $c_{ij} \leq c_{i,j+1}$ $\forall$ i and j.

Definition 2.2.2 : Matrix C has monotone property in case $D = ((d_{ij}))$ has non-decreasing rows where

$$d_{ij} = \begin{cases} c_{ij} - c_{i-1,j} & \text{if } i \geq 2, \\ 0 & \text{if } i = 1 \end{cases}$$

We now consider the assignment problem with the cost matrix C having monotone property and prove the following result.

Lemma 2.2.3 When the cost matrix $C_{r \times r}$ of the assignment problem has monotone property then X^* given below, minimizes the total cost of assignment.

$$X^* = ((x^*_{ij})) \quad \text{with} \quad x^*_{ij} = \begin{cases} 1 & \text{if } i+j = r+1 \\ 0 & \text{otherwise} \end{cases}$$

Proof : Let X be an assignment different from X^*. Suppose X and X^* have the first $\lambda-1$ columns identical, $1 \leq \lambda \leq n$ and λ is the largest such. We have then $x_{r+1-\lambda,\,\lambda} = 0$.

Now consider $x_{i_1,\lambda}$ and $x_{r+1-\lambda,\,j_1}$ where i_1, j_1 are such that $x_{i_1,\lambda} = x_{r+1-\lambda\,,j_1} = 1$.

Reassign them as follows

$$x_{r+1-\lambda,\lambda} = 1 = x_{i_1,j_1} \quad \text{and}$$

$$x_{i_1,\lambda} = 0 = x_{r+1-\lambda\,,j_1}$$

with the remaining x_{ij}'s unaltered. Let the new assignment be X'. Now,

$$C(X) - C(X') = C_{i_1,\lambda} + C_{r+1-\lambda\,,j_1} - C_{r+1-\lambda\,,\lambda} - C_{i_1 j_1}$$

$$= (C_{r+1-\lambda,j_1} - C_{i_1,j_1}) - (C_{r+1-\lambda,\lambda} - C_{i_1,\lambda}).$$

Observe, j_1 has to be greater than λ, as otherwise our assumption that X and X^* agree upto $\lambda-1^{st}$ column will be contradicted, as $x^*_{r+1-\lambda\,,\,j_1} = 0$ and $x_{r+1-\lambda,j_1} = 1$. Similarly, i_1 has to be less than $r+1-\lambda$. Since C has monotone property,

$$C_{r+1-\lambda,\,j_1} - C_{i_1,j_1} \geq C_{r+1-\lambda,\lambda} - C_{i_1,\lambda}\,.$$

Thus $C(X') \leq C(X)$ and X' agrees with X^* upto the λ^{th} column. Repeating this argument X^* is shown to be optimal. Hence the lemma follows.

Coming back to the tardiness problem,

$$\text{let } C_i' = \begin{cases} t(\sigma,m) + \sum_{s=1}^{i} p^{(s)} & \text{for } i = 1,\ldots,r-1 \\ L(\sigma) & \text{for } i = r. \end{cases}$$

With $r = n-q$ and $p^{(1)} \leq \ldots \leq p^{(r)}$, the ordered processing times on the last machine for jobs in $N-\sigma$, where σ is a partial sequence with q jobs.

Let $(q_1 \ldots q_r)$, be a sequence of the jobs in $N - \sigma$, such that $d_{q_1} - p_{mq_1} \geq \ldots \geq d_{q_r} - p_{mq_r}$.

Let $(s_1 \ldots s_r)$ be a sequence of the jobs in $N - \sigma$, such that $u_{s_1}(x) \leq \ldots \leq u_{s_r}(x)$ for all $x \in [0, \beta]$.

Let $(t_1 \ldots t_r)$ be a sequence of the jobs in $N - \sigma$, such that $d_{t_1} \geq \ldots \geq d_{t_r}$.

Let $y_{ij} = \text{Max}[C_i' - d_{t_j}, 0]$

and $z_{ij} = \text{Max}[C_{i-1}' + p_{mq_j} - d_{q_j}, 0]$

First we observe that for any sequence $(j_1 \ldots j_r)$ of the jobs in $N - \sigma$,

$$T_{j_v} \geq y_{v\lambda} \quad \text{for } \lambda \text{ such that } t_\lambda = j_v$$

$$\text{for } 1 \leq v \leq r,$$

$$\text{also } T_{j_v} \geq z_{vk} \quad \text{for } k \text{ such that } q_k = j_v$$

$$\text{for } 1 \leq v \leq r.$$

Let $A_{r\times r} = ((a_{ij}))$ where $a_{ij} = u_{s_j}(y_{i1})$,

$B_{r\times r} = ((b_{ij}))$ where $b_{ij} = u_{s_1}(y_{ij})$,

$C_{r\times r} = ((c_{ij}))$ where $c_{ij} = u_{t_j}(y_{ij})$

and $E_{r\times r} = ((e_{ij}))$ where $e_{ij} = u_{q_j}(z_{ij})$.

We have, from the definition of y_{ij} and z_{ij},

$$\text{(i)} \quad y_{ij} \leq y_{ij+1} \quad \text{and} \quad y_{ij} \leq y_{i+1,j}$$

$$\text{and (ii)} \quad z_{ij} \leq z_{ij+1} \quad \text{and} \quad z_{ij} \leq z_{i+1,j} \qquad \ldots\ldots 2.2.3$$

Moreover, if u_{s_1} is convex, then

$$u_{s_1}(y_{i+1,j}) - u_{s_1}(y_{ij}) \leq u_{s_1}(y_{i+1,j+1}) - u_{s_1}(y_{i,j+1})$$

because of the monotonicity property of convex functions.

In general, the matrix C, may not be monotone. The matrix B is monotone if u_{s_1} is convex. The matrix A is

monotone as u_j's satisfy additional assumptions 2.2.2. Similarly, in general, the matrx E may not be monotone. However, we have the following proposition. Let C and E be as defined above.

Proposition 2.2.1 : (a) If $t_j = s_j$ for all j, then the matrix C is monotone.

(b) If $q_j = s_j$ for all j, then the matrix E is monotone.

The proof of the proposition follows from 2.2.3 and the additional properties of u_j's namely assumptions 2.2.2.

Let u_A, u_B, u_C and u_E be the optimal cost of assignment with the cost matrix A,B,C and E respectively.

Thus, we have u_A, u_B, U_c and u_E as lower bounds for the cost of tardiness for any sequence of the jobs in $N - \sigma$.

We have,

$$u_A = \sum_{j=1}^{r} u_{s_j}(y_{r+1-j,1}) \quad \text{and if } u_{s_1} \text{ is convex}$$

then $$u_B = \sum_{j=1}^{r} u_{s_1}(y_{r+1-j,j})$$

using the monotone property of the matrix and lemma 2.2.3.

If an earliest due date (EDD) sequence is also a largest penalty rate (LPR) sequence then using the proposition 2.2.1,

we obtain another lower bound

$$u_c = \sum_{j=1}^{r} u_{s_j}(y_{r+1-j,j})$$

Similarly if a sequence $(s_1 \ldots s_r)$ is such that

$$u_{s_1} \leq \ldots \leq u_{s_r} \quad \text{and},$$

$$d_{s_1} - p_{s_1} \geq \ldots \geq d_{s_r} - p_{s_r} \quad \text{then using}$$

the proposition 2.2.1, we have the lower bound,

$$u_E = \sum_{j=1}^{r} u_{s_j}(z_{r+1-j,j}).$$

Moreover, when $m = 1$, we get using the proposition 2.2.1, the following lemma, which is a generalization of a result noted by Hamilton Emmons (1969).

Consider the (n/1/F/u(T)) problem with nondecreasing functions u_j's satisfying the additional assumptions 2.2.2, made in the beginning of this section. Then we have the following lemma.

Lemma 2.2.4 : A shortest processing time (SPT) sequence is optimal in case it is identical with an earliest due date(EDD) sequence and a largest penalty rate (LPR) sequence.

Remarks

(1) Note that, if a SPT sequence is such that $d_1 - p_1 \leq \ldots \leq d_r - p_r$ then the SPT sequence is identical with an EDD sequence.

(2) Shwimer (1972) has proved the lemma for linear penalty functions. But his claim about the non-linear u_j's is not correct in the light of the special cases of $u(T)$ discussed in this section and the following example.

Example 2.2.2

j	p_j	d_j
1	3	2
2	4	5
3	5	7

with $u_1(x) = \begin{cases} 10x, & 1 \geq x \geq 0 \\ 10, & 10 \geq x \geq 1 \end{cases}$

$u_2(x) = \begin{cases} 2x, & 5 \geq x \geq 0 \\ 10, & 10 \geq x \geq 5 \end{cases}$

and $u_3(x) = x, \quad 10 \geq x \geq 0$

It is easy to see that the LPR sequence is (1,2,3) as $u_1(x) \geq u_2(x) \geq u_3(x)$ for every x such that $0 \leq x \leq 10$. But these u_j's do not satisfy the assumption (ii) in 2.2.2. Further, the LPR sequence, the SPT sequence and the EDD sequence are the same, namely (1,2,3). But (1,2,3) is not optimal, since,

$$C(1,2,3) = u_1(1) + u_2(2) + u_3(5)$$
$$= 19$$

and $$C(2,3,1) = u_2(0) + u_3(2) + u_1(10)$$
$$= 12.$$

2.3 Some Special Cases of $(n/2/F/C_u)$ problem

2.3.1 Introduction :

Ignall and Schrage (1965) discuss the problem $(n/2/F/\overline{C})$ and give a Branch and Bound algorithm to solve the problem. They make use of two special cases of the problem to develop a lower bound. We give algorithms to solve the $(n/2/F/C_u)$ problem when the processing times satisfy either

$\max_j A_j \leq \min_j B_j$ or $\min_j A_j \geq \max_j B_j$. It is shown that an assertion made by Ignall and Schrage (1965) in this connection is incorrect. We take A_j's and B_j's to denote the processing times for machines A and B respectively.

2.3.2 Notations and the Special Cases

As earlier, let X_i be the idle time on the last machine, namely B, immediately before processing the job j_i, when the sequence considered is $p = (j_1 \ldots j_n)$.

Now

$$C_{j_i} = \sum_{r=1}^{i} X_r + \sum_{r=1}^{i} B_{j_r} \quad , \quad 1 \leq i \leq n \ .$$

From Johnson's (1954) paper, we have

$$\sum_{r=1}^{i} X_r = \max_{1 \leq r \leq i} K_r \qquad \text{where}$$

$$K_r = \sum_{s=1}^{r} A_{j_s} - \sum_{s=1}^{r-1} B_{j_s}$$

The problem is to minimise $C(p) = \sum_{i=1}^{n} u_{j_i} C_{j_i}$ over the set of all permutations of (1 2 ... n).

Case I $\quad \max_j A_j \leq \min_j B_j$

In this case, we have

$$K_i \geq K_{i+1} \qquad \text{for all } 1 \leq i \leq n-1$$

Therefore, $C_{j_i} = K_1 + \sum_{r=1}^{i} B_{j_r} = A_{j_1} + \sum_{r=1}^{i} B_{j_r}$

and $$\sum_{i=1}^{n} u_{j_i} C_{j_i} = \sum_{i=1}^{n} A_{j_1} u_{j_i} + \sum_{i=1}^{n} u_{j_i} \sum_{r=1}^{i} B_{j_r}$$

$$= \sum_{i=1}^{n} (A_{j_1} + B_{j_1}) u_{j_i} + \sum_{i=2}^{n} u_{j_i} \sum_{r=2}^{i} B_{j_r}$$

Mc Naughton (1959) has shown that $\sum_{i=1}^{q} u_{j_i} \sum_{r=1}^{i} p_{j_r}$ is minimum when $(s_1 \ldots s_q)$ is such that

$$p_{s_1}/u_{s_1} \leq p_{s_2}/u_{s_2} \ldots \leq p_{s_q}/u_{s_q} \qquad \ldots 2.3.1$$

Using this sequence called Mc Naughton's sequence, with $p_j = B_j$ we can minimise $\sum_{i=2}^{n} u_{j_i} \sum_{r=2}^{i} B_{j_r}$. Let P_j be

the set of all permutations with $j_1 = j$. For $P \in \underline{P}_j$, $\sum_{i=1}^{n} (A_{j_1} + B_{j_1}) u_{j_i}$ is a constant. Let the Mc Naughton's sequence of the (n-1) jobs other than j, corresponding to fixing job $j_1 = j$ be S_j. Let

$$D_j = \sum_{i=1}^{n} (A_j + B_j) u_{j_i} + R_j$$

where R_j is the cost corresponding to S_j.

$$\text{Then} \quad \min_{p} C(p) = \min_{1 \le j \le n} \left\{ \min_{p \in \underline{P}_j} C(p) \right\}$$

$$= \min_{1 \le j \le n} D_j .$$

Let j_o be such that $D_{j_o} = \min_{1 \le j \le n} D_j$. Then an optimal sequence for the problem is given by (j_o, S_{j_o}) with optimal cost D_{j_o}.

We note that the assertion made by Ignall and Schrage (1965) that $(j_1 \ldots j_n)$ such that $B_{j_1} \le \ldots \le B_{j_n}$ is optimal for $(n/2/F/C_u)$ problem with $u_j = \frac{1}{n}$ for all j, is incorrect. A counter example is given below.

Example 2.3.1 : Consider the $(4/2/F/\bar{C})$ with the data given in the following table.

j	A_j	B_j
1	3	2
2	2	3
3	3	4
4	2	5

We have, $\bar{C}(1,2,3,4) = \frac{1}{4}(2+5+9+14) + \frac{1}{4}(3 \times 4)$

$= \frac{42}{4}$

The sequence (1,2,3,4) is optimal according to the assertion made by Ignall and Schrage (1965). But we find

$$\bar{C}(2,1,3,4) = \frac{1}{4}(3+5+9+14) + \frac{1}{4}(2 \times 4)$$

$$= \frac{39}{4} < \bar{C}(1,2,3,4) = \frac{42}{4}$$

Case II $\quad \min_j A_j \geq \max_j B_j$

Under this assumption, we have

$$K_i \leq K_{i+1} \quad \text{for all} \quad 1 \leq i \leq n-1.$$

Therefore,

$$C_{j_i} = K_i + \sum_{r=1}^{i} B_{j_r}$$

$$= \sum_{r=1}^{i} A_{j_r} - \sum_{r=1}^{i-1} B_{j_r} + \sum_{r=1}^{i} B_{j_r}$$

$$= \sum_{r=1}^{i} A_{j_r} + B_{j_i}$$

and $C(p) = \sum_{i=1}^{n} u_{j_i} C_{j_i} = \sum_{i=1}^{n} u_{j_i} \sum_{r=1}^{i} A_{j_r} + \sum_{i=1}^{n} B_{j_i} u_{j_i}$.

Thus $C(p)$ is minimised when $\sum_{i=1}^{n} u_{j_i} \sum_{r=1}^{i} A_{j_r}$ is minimised as $\sum_{i=1}^{n} B_{j_i} u_{j_i}$ is a constant. We find a Mc Naughton's sequence satisfying 2.3.1 with $p_j = A_j$ for all j, $1 \leq j \leq n$, to minimise $C(p)$.

Thus, a sequence $(j_1 \ldots j_n)$ satisfying

$$A_{j_1}/u_{j_i} \leq A_{j_2}/u_{j_2} \leq \ldots \leq A_{j_n}/u_{j_n}$$

is an optimal sequence for the problem.

2.4 Some Special Cases of (n/m/F/f) problem with f a Regular Measure of Performance

2.4.1 Introduction

In chapter 1, we mentioned that the theorems 1.1.1 and 1.1.2, assuming BCD and FCD conditions can be proved for more general objectives as well. That task is taken up in this section. In this section we make the usual assumptions A_1 through A_8. We consider the (n/m/F/f) problem with f a regular measure of performance. Maxwell and others (1967) define a regular measure of performance as follows.

A function $f : R^n \rightarrow R$ is said to be a regular measure of performance in case for $C, C' \in R^n$ if $C_i \leq C'_i$ for all i, then we have $f(C) \leq f(C')$. Almost all objective functions commonly considered in sequencing literature are regular measures of performance. For example, F_{max}, C_u, number of late jobs, $u(T)$ as given in section 2.2 are regular measures of performance. Further, under the usual assumptions, while solving a (n/m/F/f) problem with a regular measure of performance, it is sufficient to consider permutation schedules. Therefore one can restrict one's search for an optimal schedule to the set of all permutations of (1 2 ... n) [See Maxwell and others (1967)]. As only this property of a regular measure of performance is used in the proofs of the theorems 2.4.1 and 2.4.2, the theorems hold good for any f with this property.

2.4.2 Main Theorems :

We need the following definition in theorem 2.4.1.

Define $f_j(x) = f(x')$ where $x'_i = x_i + a_j$ with

$$a_j = \sum_{r=1}^{k-1} p_{rj} \text{ , for } x \in R^n .$$

Theorem 2.4.1 : If there exists a k such that the BCD conditions of chapter 1 hold for machine M_k then the corresponding (n/m/F/f) problem with f a regular measure of performance, reduces to that of solving n, $(n/m-k+1/F/f_j)$ problems involving the last $m-k+1$ machines with the restriction that job j is fixed to be the first job in the j^{th} problem, and then selecting the 'best' (j_o, S_{j_o}) sequence wher (j, S_j) is an optimal sequence for the j^{th} problem.

Proof : From Lemma 1.1.1 we have for a $p = (j_1 \ldots j_n)$

$$t((j_1 \ldots j_i), k) = \sum_{r=1}^{k-1} p_{rj_1} + \sum_{s=1}^{i} p_{kj_s}, \quad 1 \leq i \leq n .$$

Let $\underline{P}_j$ be the set of sequences in which $j_1 = j$. Let $\underline{M}_i = M_{k+i-1}$ for $1 \leq i \leq m-k+1$. Consider the problem with the $m-k+1$ machines $\underline{M}_i$ and the sequences belonging to $\underline{P}_j$. Let $\underline{C}_s$ be the completion time of job s on the machine $\underline{M}_{m-k+1}$.

Then for $p \in \underline{P}_j$,

$$C_{j_i} = t((j_1 \ldots j_i), m) = \underline{C}_{j_i} + \sum_{r=1}^{k-1} p_{rj}$$

for all $1 \leq i \leq n$.

From the definition of f_j it is clear that an optimal solution to (n/m/F/f) problem over $\underline{P}_j$ can be found by finding an optimal solution to the $(n/m-k+1/F/f_j)$ problem with machines $\underline{M}_i$ over $\underline{P}_j$.

Now,

$$\underset{p}{\text{Min}}\ f = \underset{1\le j\le n}{\text{Min}} \left\{ \underset{p\in \underline{P}_j}{\text{Min}}\ f_j \right\}$$

and the theorem follows.

Next we consider the case when FCD conditions hold for machine M_k.

Let $f'(x) = f(x + a)$ where $a = (a_1 \ldots a_n)$ with

$$a_j = \sum_{r=k+1}^{m} p_{rj}, \text{ for } x \in R^n.$$

Theorem 2.4.2 : If there exists a k such that FCD conditions hold for machine M_k, then the corresponding (n/m/F/f) problem with f a regular measure of performance, reduces to that of solving a $(n/k/F/f')$ problem, involving the first k machines, where the objective function f' is as defined above.

Proof : From Lemma 1.1.2 for any sequence $p = (j_1 \ldots j_n)$, we have,

$$t((j_1 \ldots j_i), r) = t((j_1 \ldots j_i), r-1) + p_{rj_i}$$

for all r such that $k+1 \le r \le m$

Using this repeatedly, we get

$$C_{j_i} = t((j_1 \ldots j_i), m) = t((j_1 \ldots j_i), k) + \sum_{r=k+1}^{m} p_{rj_i}$$

Let $\underline{C}_j$ be the completion time of job j on the machine M_k in the problem with the machines $M_1 \ldots M_k$.

From the definition of f', we get

$$f(C_1 \ldots C_n) = f(\underline{C}_1 + a_1 \ldots \underline{C}_n + a_n)$$

$$= f'(\underline{C}_1 \ldots \underline{C}_n).$$

Thus Min f = Min f' and the theorem is proved.

Remarks :

1. Both the functions f_j and f' defined above are regular measures of performance when f is a regular measure of performance.

2. In theorem 1.1.2 for F_{max}, we solve n, $(n/k/F/F_{max})$ problems fixing job j as the last job in the j^{th} problem, whereas in theorem 2.4.2 we say the original problem is equivalent to a $(n/k/F/f')$ problem. As F_{max} is a regular measure of performance, one may wonder why not solve a $(n/k/F/f')$ problem instead of n, $(n/k/F/F_{max})$ problems. In fact solving those n problems is equivalent to solving a problem with $f' = \underset{i}{\text{Max}} (\underline{C}_i + a_i)$, since when job j is

fixed as the last job, $f' = (\underset{i}{\text{Max}}\ C_i) + a_j = C_j + a_j$,

2.4.3 Some Special Objective Functions

We now consider some objective functions where the application of the above theorems will lead to solving one or more problems with a single machine. In these cases efficient procedures for solving the reduced problems exist.

First we consider (n/m/F/f) problems with BCD conditions holding good for machine M_m.

(A) Weighted sum of completion times.

Let $f(C) = \sum_{j=1}^{n} C_j u_j$ where $C = (C_1 \ldots C_n)$

We shall show that the following procedure solves the problem :

1. Find a Mc Naughton's sequence $(s_1 \ldots s_n)$ for the $(n/1/F/C_u)$ problem with machine M_m.

2. Calculate for each j,

$$D_j = a_j \sum_{r=1}^{n} u_r + p_{mj} \sum_{r=1}^{i-1} u_{s_r} - u_j \sum_{r=1}^{i-1} p_{ms_r}$$

$$\text{with } a_j = \sum_{1}^{m-1} p_{rj}$$

and find j_o for which D_j is minimised.

3. Then $(j_o, s_1 \cdots s_{i_o-1}, s_{i_o+1} \cdots s_n)$ is an optimal sequence for the problem where $j_o = s_{i_o}$. Also the optimum value of f is $W + D_{j_o}$ when W is the weighted sum of completion time corresponding to $(s_1 \cdots s_n)$.

According to theorem 2.4.1, the problem is solved by solving n, $(n/1/F/f_j)$ problems with machine M_m and $j_1 = j$ in the j^{th} problem.

Now,

$$f_j(\underline{C}) = f(\underline{C}') \quad \text{where} \quad \underline{C}'_i = \underline{C}_i + a_j \quad \text{and} \quad a_j = \sum_{r=1}^{m-1} p_{rj}$$

$$= \sum_{i=1}^{n} u_i \, (\underline{C}_i + a_j)$$

$$= \sum_{i=1}^{n} u_i \, \underline{C}_i + \sum_{i=1}^{n} u_i \, a_j$$

So minimising f_j is equivalent to minimising $\sum_{i=1}^{n} u_i \, \underline{C}_i$, fixing $j_1 = j$. Consider any $p = (j_1 \cdots j_n) \in \underline{P}_j$

$$\sum_{i=1}^{n} u_{j_i} \, \underline{C}_{j_i} = \sum_{i=1}^{n} u_{j_i} \, p_{mj} + \sum_{i=2}^{n} u_{j_i} \left[\sum_{r=2}^{i} p_{mj_r} \right]$$

Thus, minimising f_j is the same as minimising

$$\sum_{i=2}^{n} u_{j_i} \sum_{r=2}^{i} p_{mj_r} \quad \text{as} \quad \sum_{i=1}^{n} u_{j_i} \, p_{mj} \quad \text{is a constant.}$$

And we know that Mc Naughton's sequence of the jobs other than j minimises $\sum_{i=2}^{n} u_{j_i} \sum_{r=2}^{i} p_{mj_r}$. Let the minimum here be R_j and the corresponding sequence S_j.

Therefore, an optimal sequence for the problem is given by the 'best' (j_o, S_{j_o}) sequence. Further, suppose $(s_1 \ldots s_n)$ is the Mc Naughton's sequence (given by 2.3.1) for all the n jobs on the machine M_m. If job $j = s_i$, then the minimum weighted cost for the problem involving the jobs other than j is given by

$$R_j = W - \left(\sum_{r=i}^{n} p_{mj} u_{s_r} + \sum_{r=1}^{i-1} p_{ms_r} u_j \right)$$

where W is the weighted cost corresponding to $(s_1 \ldots s_n)$.

Also $S_j = (s_1 \ldots s_{i-1} s_{i+1} \ldots s_n)$.

Thus the optimal value of f_j is

$$a_j \sum_{r=1}^{n} u_r + \sum_{r=1}^{n} p_{mj} u_r + R_j.$$

Simplifying we find, the 'best' (j_o, S_{j_o}) sequence is same as $(j_o, s_1, \ldots, s_{i_o-1}, s_{i_o+1}, \ldots, s_n)$ where $j_o = s_{i_o}$ and $D_{j_o} = \min_{1 \le j \le n} D_j$.

(B) Number of late jobs

$$f(C) = \sum_{i=1}^{n} x_i \qquad \text{where}$$

$$x_i = \begin{cases} 1 & \text{if } C_i - d_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

By theorem 2.4.1, the problem is solved by solving n, $(n/1/F/f_j)$ problems with machine M_m and $j_1 = j$ in the j^{th} problem.

Now
$$f_j(\underline{C}) = f(\underline{C}') \text{ where } \underline{C}'_i = \underline{C}_i + a_j \text{ with } a_j = \sum_{r=1}^{m-1} p_{rj}$$

$$= \sum_{i=1}^{n} x_i \text{ where } x_i = \begin{cases} 1 & \text{if } \underline{C}_i + a_j - d_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

Thus solving the j^{th} problem is same as solving a $(n/1/F/f)$ problem, over $p \in \underline{P}_j$ with machine M_m and due date $d_i - a_j$ for job i. Moore (1968) has given a simple rule for solving the $(n/1/F/f)$ problem with f = number of late jobs. This appears in section 2.7. Let S_j be a sequence of the $n-1$ jobs other than j obtained using Moore's rule for the j^{th} problem. Then f is minimised over all permutations, when j_o is chosen such that (j_o, S_{j_o}) is the "best" among (j, S_j) for all j.

We note that one needs to consider the j^{th} problem, only if $p_{mj} + a_j \leq d_j$.

(C) Maximum $u_i(T_i)$

$$f(C) = \max_{1 \leq i \leq n} u_i(T_i) \quad \text{where}$$

$$T_i = \max(C_i - d_i, 0)$$

with u_i a nondecreasing function of T_i for every i.

By theorem 2.4.1, the problem is solved by solving n $(n/1/F/f_j)$ problems with machine M_m and $j_1 = j$ in the j^{th} problem.

Now

$$f_j(\underline{C}) = f(\underline{C}') \quad \text{where} \quad \underline{C}'_i = \underline{C}_i + a_j \text{ with } a_j = \sum_{r=1}^{m-1} p_{rj}$$

$$= \max_{1 \leq i \leq n} u_i(T'_i) \quad \text{and} \quad T'_i = \max(\underline{C}_i + a_j - d_i, 0).$$

Thus solving the j^{th} problem is same as solving a $(n/1/F/f)$ problem with machine M_m and due date $d_i - a_j$ for job i, over $p \in \underline{P}_j$. Lawler (1973) has developed a simple algorithm for solving $(n/1/F/f)$ problem where $f = \max u_i(T_i)$. Let $t = \sum_{r=1}^{n} p_{mr}$. We find an optimal sequence for the j^{th} problem as follows. Set $J = N - \{j\}$, $f^* = u_j(T)$

where $T = \max(p_{mj} + a_j - d_j, 0)$. Also set $r = n$.

Step 1 : Find j_r such that $u_{j_r}(T^!_{j_r}) = \underset{i \in J}{\text{Min}}\ u_i(T^!_i)$

where $T^!_i = \text{Max}(t - d_i + a_j,\ 0)$

Set $f^* = \text{Max}[u_{j_r}(T^!_{j_r}),\ f^*]$

Set $J = J - \{j_r\}$. Go to Step 2.

Step 2 : If $J = \phi$ go to Step 3.

Otherwise, set $t = t - p_{mj_r}$

and $r = r-1$, go to Step 1.

Step 3 : $S_j (= (j_2 \ldots j_n)$ and (j, S_j) gives an optimal sequence for the j^{th} problem, with f^* as the optimal f value. Stop.

Using this algorithm the j^{th} problem can be solved for each j and the ''best'' (j_o, S_{j_o}) is an optimal sequence for the problem.

(D) Sum of the absolute deviations of completion times

$$f(C) = \sum_{i<\lambda} |C_i - C_\lambda|$$

It is easy to see that this objective function is not a regular measure of performance. But still we can use the

theorem 2.4.1 as permutation schedules are sufficient to consider, in this case.

Thus the problem reduces to that of solving n, $(n/1/F/f_j)$ problems with machine M_m and $j_1 = j$ in the j^{th} problem.

Now

$$f_j(\underline{C}) = f(\underline{C}') \quad \text{where} \quad \underline{C}_i' = \underline{C}_i + a_j \quad \text{with} \quad a_j = \sum_{r=1}^{m-1} p_{rj}$$

$$= \sum_{i<\ell} |\underline{C}_i - \underline{C}_\ell| = f(\underline{C})$$

So, the j^{th} problem can be solved by solving a $(n/1/F/f)$ problem on the machine M_m. Further, since $f_j(\underline{C})$ does not depend on j, solving $(n/m/F/f)$ problem boils down to solving a $(n/1/F/f)$ problem on the machine M_m over all permutations. Elmaghraby (1968a) has considered this problem of minimising sum of the absolute deviations of completion times and gives the following rule to obtain an optimal sequence.

Step 1 : Rename the jobs such that

$$p_{m1} \geq p_{m2} \geq \cdots \geq p_{mn}$$

Step 2. Fix $s_1 = 1$.

$$\text{Fix} \quad s_{i+1} = \begin{cases} 2+2(i-1) & \text{for } 1 \leq i \leq [\frac{n-1}{2}] \\ 3+2(n-i-1) & \text{for } n - [\frac{n-1}{2}] \leq i \leq n-1 \end{cases}$$

If n is odd go to Step 3. Otherwise fix $s_{[\frac{n-1}{2}]+2} = n$. Go to Step 3.

Step 3 : $(s_1 \ldots s_n)$ is optimal. Stop.

Now we consider some applications of the theorem 2.4.2. From now on we consider (n/m/F/f) problems with FCD conditions holding good for machine M_1.

(E) Weighted sum of completion times

$$f(C) = \sum_{i=1}^{n} u_i C_i$$

The problem reduces to that of solving a (n/1/F/f') problem with machine M_1. We have

$$f'(\underline{C}) = f(\underline{C} + a) \quad \text{where} \quad a = (a_1 \ldots a_n) \text{ and } a_j = \sum_{r=2}^{m} p_{rj}$$

$$= \sum_{i=1}^{n} u_i (\underline{C}_i + a_i)$$

$$= \sum_{i=1}^{n} u_i (\underline{C}_i) + \sum_{i=1}^{n} u_i a_i$$

$$= f(\underline{C}) + \text{constant.}$$

Hence, we need to solve a (n/1/F/f) problem with machine M_1. Mc Naughton's rule gives an optimal solution.

(F) Number of late jobs

f(C) is as defined in (B).

The problem is solved by solving a (n/1/F/f') problem with machine M_1. Now

$$f'(\underline{C}) = f(\underline{C} + a) \text{ with } a = (a_1 \ldots a_n) \text{ and } a_j = \sum_{r=2}^{m} p_{rj}$$

$$= \sum_{i=1}^{n} x_i \text{ where } x_i = \begin{cases} 1 & \text{if } \underline{C}_i + a_i - d_i > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Thus we need to solve a (n/1/F/f) problem with due dates $d_i - a_i$ and machine M_1. Moore's rule (1968) yields an optimal solution to the problem.

(G) Maximum $u_i(T_i)$

$$f(C) = \text{Max } u_i(T_i) \text{ with } T_i = \text{Max}(C_i - d_i, 0).$$

The problem is equivalent to a (n/1/F/f') problem with machine M_1 and

$$f'(\underline{C}) = f(\underline{C} + a) \text{ with } a = (a_1 \ldots a_n) \text{ and } a_j = \sum_{r=2}^{m} p_{rj}$$

$$= \text{Max}_{1 \le i \le n} u_i(T_i') \text{ with } T_i' = \text{Max}(\underline{C}_i + a_i - d_i, 0)$$

Then solving the problem is same as solving a (n/1/F/f) problem with $d_i - a_i$ as due dates and machine M_1.

Remarks

1. When f is as in case D (and FCD conditions hold for machine M_1) the problem reduces to that of solving a $(n/1/F/f')$ problem with machine M_1 and

$$f'(\underline{C}) = f(\underline{C} + a) \quad \text{with} \quad a = (a_1 \ldots a_n) \text{ and } a_j = \sum_{r=2}^{m} p_{rj}$$

$$= \sum_{i<\ell} |\underline{C}_i + a_i - \underline{C}_\ell - a_\ell|$$

There is no simple rule available to get an optimal solution in this case.

2. It might be possible to use repeatedly the theorems 2.4.1 and 2.4.2 and reduce the number of machines in the problems to be solved considerably. For example, if for machine M_k, FCD conditions hold, then we have a $(n/k/F/f')$ problem to be solved with first k machines and $f'(x) = f(x+a)$ where $a_i = \sum_{r=k+1}^{m} p_{ri}$ and $a = (a_1 \ldots, a_n)$. Now, further if BCD conditions hold for machine M_k, then it reduces to solving n, $(n/1/F/f'_j)$ problems with machine M_k and $f'_j(x) = f'(x')$ where $x'_i = x_i + b_j$ and $b_j = \sum_{r=1}^{k-1} p_{rj}$. That is, $f'_j(x) = f(x + \omega_j)$ where $\omega_j = (a_1 + b_j, a_2 + b_j, \ldots, a_n + b_j)$. Thus solving

$(n/m/F/f)$ problem reduces to that of solving n, $(n/1/F/f_j')$ problems with machine M_k.

3. The cases 1 and 2 considered in section 2.3 are particular cases of cases A and E respectively with $m = 2$.

2.5 Minimisation of Weighted Tardiness with a Single Machine

2.5.0 Introduction :

Elmaghraby (1968) has considered the $(n/1/F/T_u)$ problem and has given a partial search method after converting the problem into that of determining the shortest path in a directed acyclic network. The branching for optimal sequence is done by fixing the last job at any stage, from among the remaining jobs. He also gives a simple lower bound for the cost of tardiness for the remaining jobs. Hamilton Emmons (1969) has considered the $(n/1/F/\overline{T})$ problem and has proved theorems that establish the relative order in which a pair of jobs is processed in an optimal schedule. Srinivasan (1971) has used these results of Hamilton Emmons (1969) along with the dynamic programming method to get a hybrid algorithm for solving $(n/1/F/\overline{T})$ problem. Shwimer (1972) gives a Branch and Bound procedure for the $(n/1/F/T_u)$ problem.

In this section, we consider the $(n/1/F/T_u)$ problem under usual assumptions. The results of Hamilton Emmons (1969) are extended to this problem in section 2.5.1.

In section 2.5.2 we specialize the algorithm given in section 2.2 to the $(n/1/F/T_u)$ problem and incorporate the results of section 2.5.1 along with the improved bound discussed in section 2.2, for the case of the linear penalty function T_u.

2.5.1 The Relative Order of Jobs in an Optimal Sequence

The method of precedences introduced by Hamilton Emmons (1969) is as follows. Initially there are no precedence relations known among jobs. At any stage suppose it is known that i_p precedes j_p $(i_p \leftarrow j_p)$ for $p = 1,\ldots,\lambda$ i.e., there is an optimal sequence in which i_p appears before j_p for $p = 1,\ldots,\lambda$. It is clear that this precedence relation between jobs is transitive. Let B_k be the set of jobs which precede k, i.e. $B_k = \{i_p : 1 \leq p \leq \lambda, j_p = k\}$ and A_k be the set of jobs which succeed k i.e. $A_k = \{j_p : 1 \leq p \leq \lambda, i_p = k\}$. At any stage we define the precedence matrix

$$M = ((m_{ij})) \text{ with } m_{ij} = \begin{cases} 1 & \text{if } i \text{ precedes } j \\ 0 & \text{otherwise.} \end{cases}$$

Then by definition $A_k = \{j/m_{kj} = 1\}$ and $B_k = \{j/m_{jk} = 1\}$.

We denote N-A by A^c. Let the jobs be numbered such that $p_1 \leq \ldots \leq p_n$. Without loss of generality let p_i, u_i be positive for all i. [Note that we are assuming, p_i, $u_i \geq 0$ for all i]

Let $p(A) = \sum_{i \in A} p_i$ for any subset A of N.

Let $p(A_j^c) = p'$.

Definition 2.5.1 : Let $\delta_{kj} = \underset{x,y}{\text{Min}}\ f(x, y)$ subject to

$$y + p_j + p_k \leq x \leq p' \quad \text{and} \quad p(B_k) \leq y,$$

where

$$f(x, y) = \frac{x - \text{Max}(y + p_j, d_j)}{x - \text{Max}(y + p_k, d_k)}$$

Without loss of generality let $d_i < p(N)$ for all i. We now prove a theorem which gives a precedence relation $j \leftarrow k$ under certain conditions.

Theorem 2.5.1 : For any two job j and k with $j < k$, if

(i) $d_j \leq \text{Max}[p(B_k) + p_k, d_k]$

and (ii) $u_j \geq u_k$ or $1 < u_k/u_j \leq \delta_{kj}$

then $j \leftarrow k$.

Proof : Consider any two jobs j and k which satisfy the assumptions made by the theorem and any sequence S in which jobs in B_i appear before i for all i and job k appears before job j. Let S_k and C_j be the starting time of job k and completion time of job j, respectively, in S. Consider S' obtained from S by interchanging jobs j and k. We shall show that the weighted tardiness for S' is less than that of S. It is sufficient to consider the net change in the weighted tardiness for jobs j and k, since the tardiness of each job occuring between j and k in S is more than the corresponding tardiness in S' and the tardiness of all other jobs remains the same. Let Δ_i denote the resulting change in the weighted tardiness for job i.

Now Condition (i) of the theorem implies

$$d_j \leq \text{Max}(S_k + p_k,\ d_k) \quad \text{since} \quad p(B_k) \leq S_k$$

and $S_k + p_j \leq S_k + p_k$ as $p_j \leq p_k$.

So, $$\text{Max}(S_k + p_j,\ d_j) \leq \text{Max}(S_k + p_k,\ d_k) \quad \ldots \quad 2.5.1$$

Case 1 $d_j \leq d_k$ and $d_k < C_j$

Now

$$\Delta_j = u_j\ [\text{Max}\ (S_k + p_j,\ d_j) - C_j]$$

$$\Delta_k = u_k\ [C_j - \text{Max}(S_k + p_k,\ d_k)]$$

Therefore

$$\Delta_j + \Delta_k = u_k[C_j - \text{Max}(S_k+p_k, d_k)] - u_j[C_j - \text{Max}(S_j+p_j, d_j)]$$

By hypothesis, either $u_j \geq u_k$ or $1 < u_k/u_j \leq \delta_{kj}$. First let $u_j \geq u_k$. Then, $\Delta_j + \Delta_k \leq 0$ from 2.5.1, which implies that the interchange of j and k leads to a decrease in the weighted tardiness.

Next let $1 < u_k/u_j \leq \delta_{kj}$.

We note that

$$\Delta_j + \Delta_k \leq 0 \quad \text{if}$$

$$u_k/u_j \leq \frac{C_j - \text{Max}(S_k + p_j, d_j)}{C_j - \text{Max}(S_k + p_k, d_k)}$$

Also we have,

(i) $p(B_k) \leq S_k$

and (ii) $S_k + p_k + p_j \leq C_j \leq p(A_j^c)$.

Now, from the definition of δ_{kj} we see,

$$\delta_{kj} \leq \frac{C_j - \text{Max}(S_k + p_j, d_j)}{C_j - \text{Max}(S_k + p_k, d_k)}$$

Since $u_k/u_j \leq \delta_{kj}$, it follows that $\Delta_j + \Delta_k \leq 0$. Hence the interchange of j and k leads to a decrease in the weighted tardiness.

Case 2 $\quad d_j \leq d_k \quad$ and $\quad C_j \leq d_k$

Now

$$\Delta_j = u_j[\text{Max}(S_k + p_j, d_j) - \text{Max}(C_j, d_j)]$$

$$\leq 0$$

since $\quad \text{Max}(C_j, d_j) \geq \text{Max}(S_k + p_j, d_j)$

And $\quad \Delta_k = u_k[\text{Max}(C_j, d_k) - \text{Max}(S_k + p_k, d_k)]$

$$= 0 \quad (\text{since} \quad d_k \geq C_j \geq S_k + p_k)$$

Therefore $\quad \Delta_j + \Delta_k \leq 0.$

Case 3 $\quad d_k < d_j$. Then by hypothesis, $d_j \leq p(B_k) + p_k$.

In this case $\quad C_j > d_j$. We have

$$\Delta_j + \Delta_k = u_k[C_j - \text{Max}(S_k + p_k, d_k)]$$

$$- u_j[C_j - \text{Max}(S_k + p_j, d_j)]$$

Now it follows that $\Delta_j + \Delta_k \leq 0$ exactly as in Case (1). Thus if S is optimal then S' is also optimal and j precedes k in S'. This completes the proof of the theorem.

The author proved the above theorem in 1971 and later discovered that Shwimer has proved a weaker theorem in 1972 which is applicable only when $u_j \geq u_k$ and condition (i) of the above theorem 2.5.1 holds. The following lemma solves

the problem of minimising $f(x, y)$ *subject to the constraints* given in definition 2.5.1.

Lemma 2.5.1 : Let condition (i) of the theorem 2.5.1 hold for some pair of jobs j and k such that $j < k$. Let (x^*, y^*) minimise $f(x, y)$ subject to the constraints given in the definition of δ_{kj} together with $x \geq d_k$. When $d_k \leq p'$, if $d_k - p_k > d_j - p_j$ then (x^*, y^*) is given by

$$(x^*, y^*) = \begin{cases} (p', p' - p_j - p_k) & \text{if } p' - p_j < d_k \\ (p', p(B_k)) & \text{if } d_k < p(B_k) + p_k \\ (p', d_k - p_k) & \text{otherwise,} \end{cases}$$

if $d_k - p_k \leq d_j - p_j$, then

$$(x^*, y^*) = (p', p(B_k)).$$

If $d_k > p'$ there is no feasible solution to the problem.

Proof : From the constraints in the definition of δ_{kj} and condition (i) of theorem 2.5.1, we get

$$\text{Max}(y + p_k, d_k) \geq \text{Max}(y + p_j, d_j).$$

Using $x \geq d_k$, we also get $x \geq \text{Max}(y + p_k, d_k)$.

Now,

$$f(x, y) = \frac{x - \text{Max}(y + p_j, d_j)}{x - \text{Max}(y + p_k, d_k)}$$

$$= 1 + \frac{\text{Max}(y + p_k, d_k) - \text{Max}(y + p_j, d_j)}{x - \text{Max}(y + p_k, d_k)} \geq 1$$

Consider the problem : $\underset{x,y}{\text{Min}} \; f(x, y)$

subject to $p(B_k) \leq y$

$$y + p_j + p_k \leq x \leq p'$$

and $d_k \leq x$

Note that in this problem, for a fixed y, $f(x, y)$ is minimum when $x = p'$. So we consider the minimization of $f(p', y)$ subject to $p(B_k) \leq y \leq p' - p_j - p_k$.

Case A $\quad d_k - p_k > d_j - p_j$

Now

$$f(p', y) = \begin{cases} \dfrac{p' - d_j}{p' - d_k} & \text{if } y \leq d_j - p_j \\ \dfrac{p' - y - p_j}{p' - d_k} & \text{if } d_j - p_j \leq y \leq d_k - p_k \\ \dfrac{p' - y - p_j}{p' - y - p_k} & \text{if } y \geq d_k - p_k \end{cases}$$

It is easy to see that (1) $f(p', y)$ is a constant when $y \leq d_j - p_j$, (2) $f(p', y)$ decreases as y increases from $d_j - p_j$ to $d_k - p_k$ and (3) $f(p', y)$ increases as y increases when $y \geq d_k - p_k$.

We consider three cases A.1, A.2 and A.3.

Case A.1 $d_k < p(B_k) + p_k$. In this case, $y > d_k - p_k$ and it is easy to see that $f(p', y)$ is minimised when $y = p(B_k)$ and

$$f(p', p(B_k)) = \frac{p' - p(B_k) - p_j}{p' - p(B_k) - p_k}.$$

Case A.2 $p(B_k) + p_k \leq d_k \leq p' - p_j$.

The unconstrained minimum of $f(p', y)$ is attained at $y = d_k - p_k$ and $(d_k - p_k) \in [p(B_k), p' - p_j - p_k]$ which is the domain of y. We get, $f(p', d_k - p_k) = \frac{p' - d_k + p_k - p_j}{p' - d_k}$.

Case A.3 $d_k > p' - p_j$.

In this case, $y \leq p' - p_j - p_k < d_k - p_k$. So it is easy to see that $f(p', y)$ is minimised at $y = p' - p_j - p_k$ and

$$f(p', p' - p_j - p_k) = \begin{cases} \frac{p' - d_j}{p' - d_k} & \text{if } p' - d_j \leq p_k \\ \frac{p_k}{p' - d_k} & \text{otherwise.} \end{cases}$$

Thus the lemma is proved in Case A.

Case B $\quad d_j - p_j \geq d_k - p_k$.

In this case

$$f(p', y) = \begin{cases} \dfrac{p' - d_j}{p' - d_k} & \text{if } y \leq d_k - p_k \\ \dfrac{p' - d_j}{p' - y - p_k} & \text{if } d_k - p_k \leq y \leq d_j - p_j \\ \dfrac{p' - y - p_j}{p' - y - p_k} & \text{if } y \geq d_j - p_j \end{cases}$$

It is easy to see that (1) $f(p', y)$ is a constant for $y \leq d_k - p_k$ and (2) $f(p', y)$ increases as y increases for $y \geq d_k - p_k$. Hence, the minimum of $f(p', y)$ is attained at $y = p(B_k)$.

Thus the proof of the lemma is complete.

The following corollaries are immediate.

Corollary 2.5.1 : If job 1 has the property

(i) $\quad d_1 \leq \text{Max}(p_i, d_i)$

and (ii) $\quad u_1 \geq u_i$ or $1 < u_i/u_1 \leq \delta_{i1}$ for all $i > 1$

then job $1 \leftarrow i$ for all $i > 1$ and the problem reduces to finding an optimal sequence in $\underline{P}_1$.

Corollary 2.5.2 : If job n has the property

(i) $\text{Max}(p_n, d_n) \geq d_i$

and (ii) $u_i \geq u_n$ or $1 < u_n/u_i \leq \delta_{ni}$

for all $i < n$

then job $i \leftarrow n$ for all $i < n$ and again the problem reduces to one with $n-1$ jobs.

Corollary 2.5.3 : The SPT sequence is optimal if it is identical with the LPR sequence and

$$d_j + p_j \leq \sum_{i=1}^{j+1} p_i \quad \forall \quad j = 1, \ldots, n-1.$$

This corollary can be proved by repeated use of corollary 2.5.1.

Definition 2.5.2 : Let $\beta_{jk} = \min_x g(x)$

subject to $p(B_k) + p_k + p_j \leq x \leq p(A_k^c)$

$$d_j \leq x$$

where $$g(x) = \frac{x - \text{Max}(x - p_j, d_k)}{x - d_j}.$$

Theorem 2.5.2 : For any two jobs j and k with $j < k$ if

(i) $d_j > \text{Max}[p(B_k) + p_k, d_k]$

(ii) $d_j + p_j \geq p(A_k^c)$

and (iii) $u_j \leq u_k$ or $1 < u_j/u_k \leq \beta_{jk}$

then $k <- j$, where β_{jk} is as defined in definition 2.5.2.

Proof : Consider any two jobs satisfying the assumptions made by the theorem and any schedule S in which jobs in B_i appear before i for all i, and job j appears before job k. Let S_j and C_k be the starting time of job j and completion time of job k, respectively, in S. Consider S' obtained from S by advancing all jobs occuring in S after job j and upto job k by p_j and starting job j at time $C_k - p_j$, the remaining jobs being unaffected, (see fig. 2.5.1).

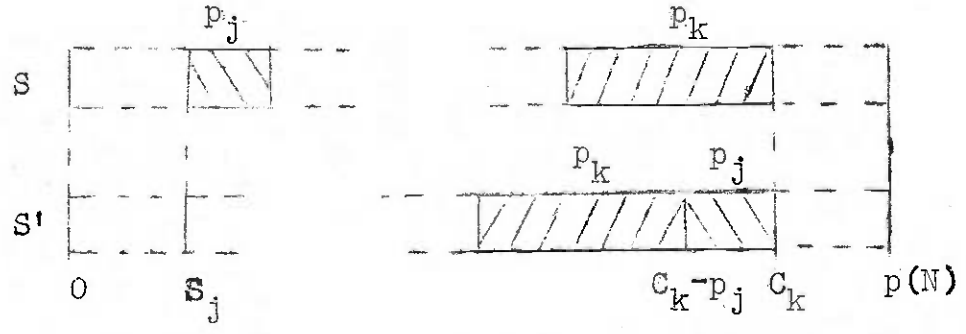


Figure 2.5.1

We shall show that the weighted tardiness for S' is less than that of S. We consider Δ_j and Δ_k (as defined in the proof of theorem 2.5.1) and show that $\Delta_j + \Delta_k \leq 0$. We have

$$C_k \leq p(A_k^c) \quad \text{and} \quad S_j \leq p(A_k^c) - p_j - p_k.$$

Case 1 $\quad C_k > d_j$

Now,

$$\Delta_k = u_k[\text{Max}(C_k - p_j, d_k) - C_k]$$

and $$\Delta_j = u_j[C_k - \text{Max}(S_j + p_j, d_j)]$$

But from Condition (ii) $d_j + p_j \geq p(A_k^c)$

$$\Rightarrow \quad d_j \geq p(A_k^c) - p_j$$

and $$S_j + p_j \leq p(A_k^c) - p_k$$

$$\leq p(A_k^c) - p_j. \quad \text{So} \quad d_j \geq S_j + p_j.$$

Therefore $\Delta_j = u_j[C_k - d_j]$ and so

$$\Delta_j + \Delta_k = u_j[C_k - d_j] - u_k[C_k - \text{Max}(C_k - p_j, d_k)]$$

Note that, $d_j \geq \text{Max}(C_k - p_j, d_k)$ because

$d_j > d_k$ from Condition (1)

and $d_j \geq p(A_k^c) - p_j \geq C_k - p_j$ from Condition (2).

Case 1.a $u_j \leq u_k$. In this case

$$\Delta_j + \Delta_k \leq 0 \quad \text{as} \quad d_j \geq \text{Max}(C_k - p_j,\ d_k).$$

Case 1.b $1 < u_j/u_k \leq \beta_{jk}$

Now, $\Delta_j + \Delta_k \leq 0$ if $1 < u_j/u_k \leq \dfrac{C_k - \text{Max}(C_k - p_j,\ d_k)}{C_k - d_j}$

The last inequality follows from the definition of β_{jk}.

Case 2 $C_k \leq d_j$. In this case $\Delta_j = 0$ and $\Delta_k \leq 0$ and hence $\Delta_j + \Delta_k \leq 0$.

The proof of the theorem is complete.

Now consider the function $g(x)$ given by definition 2.5.2, under the Conditions (i) and (ii) of the theorem. We see that $g(x)$ decreases as x increases and so $g(x)$ attains its minimum, subject to

$$p(B_k) + p_k + p_j \leq x \leq p(A_k^c) \quad \text{and} \quad x \geq d_j,$$

at $x^* = p(A_k^c)$ if $d_j \leq p(A_k^c)$. Using this x^*, β_{jk} can be calculated. If $d_j > p(A_k^c)$ there is no feasible solution to the problem.

Remarks :

(i) By taking $A_k = \phi$, in theorem 2.5.2, some corollaries similar to corollaries 2.5.1 through 2.5.3 can be proved.

(ii) Theorem 3 in Hamilton Emmons (1969) holds without any condition on u_j's and we state it here without proof :

For any two jobs j and k with $j < k$,
if $d_k \geq p(A_j^c)$, then $j <— k$.

We illustrate how these results can be used to find precedence relations among jobs.

Example 2.5.1 : Consider the $(10/1/F/T_u)$ problem for the data, given in the following table, which is obtained by adding the u_i row to an example given in Hamilton Emmons (1969).

i	1	2	3	4	5	6	7	8	9	10
p_i	6	12	16	23	32	49	61	66	80	97
d_i	25	73	31	67	32	31	57	64	15	55
u_i	25.1	20.4	25.3	22.0	22.0	18.0	18.5	19.0	19.2	19.8
$d_i - p_i$	19	61	15	44	0	-18	-4	-2	-65	42
$\text{Max}(d_i, p_i)$	25	73	31	67	32	49	61	66	80	97

We have $\sum_{i=1}^{10} p_i = 442$.

We apply the theorems 2.5.1 and 2.5.2 as often as possible to eliminate the last job at every stage. When this is no

more fesible we try and remove the first job. In this process if we are able to order all the jobs, branching is not resorted to. Otherwise we use the Branch and Bound algorithm developed in section 2.2 specialized to the single machine weighted tardiness problem. It is possible to get new precedence relations while branching from a partial σ. In this example we are able to find a complete optimal sequence without going in for branching. Also we note that by using $u_j \geq u_k$ alone in Condition (ii) of the theorem 2.5.1 no job gets fixed as either the first or the last.

Stage 1 : For job 10 we have (i) $\text{Max}(p_{10}, d_{10}) = 97 \geq d_i$ for all $i < 10$ and (ii) $u_i \geq u_{10} = 19.8$ for i=1,2,3,4 and 5. For $i = 6,7,8$ and 9, using lemma 2.5.1 we have

i	u_{10}/u_i	$\delta_{10\,i}$
6	1.100	1.139
7	1.070	1.104
8	1.042	1.089
9	1.031	1.049

Thus, using corollary 2.5.1, job 10 is last. Remove job 10 and repeat the process.

Stage 2 : Now, job 9 is last since (i) $\text{Max}(p_9, d_9)=80 \geq d_i$ for all $i < 9$ and (ii) $u_i \geq u_9 = 19.2$ for $i = 1,2,3,4$ and 5. For $i = 6,7$ and 8, we have

i	u_9/u_i	$\delta_{9\,i}$
6	1.067	1.116
7	1.038	1.071
8	1.011	1.052

Remove job 9 and repeat.

Stage 3 : We have for job 8, $\text{Max}(p_8, d_8) = 66 \geq d_i$ for $i = 1,3,5,6$ and 7 and $u_i \geq u_8 = 19.0$ for $i = 1,3,5$ and $u_8/u_7 = 1.027 > \delta_{87} = 1.025$ and $u_8/u_6 = 1.056 \leq \delta_{86} = 1.085$ using corollary 2.5.1, we have $B_8 = \{1,3,5,6\}$. With this B_8 we have $d_i \leq \text{Max}\,(p(B_8) + p_8,\ d_8) = 169$ and $u_i \geq u_8$ for $i = 2$ and 4 and

$$\delta_{87} = \frac{265 - \text{Max}(103+61,57)}{265 - \text{Max}(103 + 66,64)}$$

$$= 1.052 > u_8/u_7 = 1.027.$$

Thus, $B_8 = \{1,2,3,4,5,6,7\}$. So 8 is last. Remove job 8 and repeat.

Stage 4 : We have $\text{Max}(p_7, d_7) = 61 \geq d_i$ for $i = 1,3,5$ and 6 and $u_i \geq u_7 = 18.5$ for $i = 1,2,3,4$ and 5. Also $\delta_{76} = 1.086 > u_7/u_6 = 1.039$. Thus $B_7 = \{1,3,5,6\}$. With this B_7, $d_i \leq \text{Max}\,(103 + 61, 57) = 164$ for $i = 2$ and 4. So $B_7 = \{1,2,3,4,5,6\}$. Hence 7 is last.

Stage 5 : Similarly for job 6, initially we get $B_6 = \{1,3,5\}$. Then $d_i \leq \text{Max}\{54 + 49, 31\} = 103$ for $i = 2$ and 4. So 6 is last.

Stage 6 : For job 5, initially we get $B_5 = \{1, 3\}$. But then $\text{Max}(p(B_5) + p_5, d_5) = 54 < d_2$ and 5 cannot be removed. Thus we have the reduced problem with jobs 1 through 5 and the data obtained by omitting the last 5 columns of the given table.

Now, we try to remove the job that can precede all other jobs. We find job 1 has $d_1 = 25 \leq \text{Max}(p_i, d_i)$ for all $i > 1$ and $u_1 \geq u_i$ for $i = 2,4$ and 5.

Now, with $A_j = \{2,4,5\}$, $p(A_j^c) = 22 < 31 = d_3$. Applying theorem 3 of Hamilton Emmons (1969) $1 <- 3$. Hence job 1 is the first job. Removing it from the problem, we up date the due dates for job 2,3,4 and 5 as 67,25,61 and 26 respectively.

Stage 7 : For job 2, $d_2 = 67 \geq d_i$ *for* $i = 3, 4$ *and 5 but* $d_2 + p_2 = 79 < 83$. *So theorem 2.5.2 does not yield any result. For job 3,* $d_3 \leq Max(d_i, p_i)$ *for* $i = 4$ *and 5 and* $u_3 \geq u_4$ and u_5. So, $A_3 = \{4,5\}$. Now among jobs 2 and 3, $d_2 > d_3$; $d_2 + p_2 = 79 \geq p(A_3^c) = 28$ and $u_2 \leq u_3$ so $3 \leftarrow 2$. Thus, $A_3 = \{2,4,5\}$. Remove 3 from further consideration. Up date d_2, d_4 and d_5 as 51, 45 and 10 respectively.

Stage 8 : Now jobs 5 and 4 when compared yield, $d_4 + p_4 = 68 \geq p(A_5^c) = 67$, and $u_4 = u_5 = 22.0$. We have $A_5 = \{4\}$, $d_2 + p_2 = 63 \geq p(A_5^c) = 44$ and $u_2 = 20.4 \leq u_5$. Thus $A_5 = \{2,4\}$. So 5 is the first. Up dating d_2 and d_4 we have 19 and 13 respectively. We find $d_2 < Max(23, 13)$, $u_4/u_2 = 1.078$ and $\delta_{42} = \frac{45 - 19}{45 - 23} = \frac{26}{22} = 1.18$. So $2 \leftarrow 4$. Thus an optimal sequence is given by (1,3,5,2,4,6,7,8,9,10).

Next we consider an example in which branching is needed for obtaining an optimal sequence. The **Branch** and Bound algorithm given in section 2.2 is modified here to suit the single machine case. We fix the jobs in the reverse order, namely, the last position in a sequence is fixed in the first stage and the last but first position is fixed next and so on. Lower bounds u_A and u_B discussed under special cases of u(T)

are applicable here. But C_i' needs to be redefined since we are fixing the jobs in the reverse order. We have

$$C_i' = \sum_{s=1}^{i} p^{(s)} \quad \text{for} \quad i = 1,2,\ldots,r$$

where $p^{(1)} \leq p^{(2)} \leq \ldots \leq p^{(r)}$ are the ordered processing times of the set of all jobs in N-R.

This fixing of jobs in the reverse order is of much use when $d_j = d$ for all j. This will be discussed in detail in section 2.6. The dominance theorem given in section 2.1, when modified, is applicable in this case and has the following simple form :

Let R and R' be two reverse sequences of the same set of jobs with weighted tardiness $h(R)$ and $h(R')$ respectively, and $h(R) \leq h(R')$. Then for any reverse sequence R'' of all the jobs in N - R, we have $T_u[R'R''] \geq T_u[RR'']$ where $T_u(R'R'')$ and $T_u(RR'')$ are the weighted tardiness corresponding to (R'R'') and (RR'') respectively.

We now solve a $(7/1/F/T_u)$ problem given, in Elmaghraby (1968), as described above. The following table gives the necessary data (The jobs are renumbered according to the non-decreasing order of p_i's).

Example 2.5.2 :

i	1	2	3	4	5	6	7
p_i	1	2	3	3	4	4	5
d_i	8	6	2	5	15	17	10
u_i	1	4	1	3	3	1.5	2
$d_i - p_i$	7	4	-1	2	11	13	5
$\text{Max}(d_i, p_i)$	8	6	3	5	15	17	10

Applying the theorems 2.5.1 and 2.5.2 we get the following precedence matrix. This is done as in the previous example.

i \ j	1	2	3	4	5	6	7
1						1	
2	1				1	1	1
3							
4	1				1	1	1
5						1	
6							
7							

We find jobs 3, 6 and 7 can be last jobs. We fix them one by one as the last job and calculate the bounds using u_B given

Mr. Apurba Guha has done an elegant and quick job of duplicating the material. I thank him for that.

Finally, I thank my wife, Sujata, for cheerfully facing the inevitable inconvenience and more importantly, for her love and understanding.

°°°°°°oooo°°°°°°

in section 2.2 with C_i' defined as above. The tree generated corresponding to the problem is given in fig. 2.5.2. The optimal sequence is (2417563) with weighted tardiness 25.

We may use the precedence theorems at any node R to get further precedence relations among jobs in N-R. We note that a precedence relation $i \leftarrow j$ in N - R means that there exists a completion of R in which i is processed before j and which has the least weighted tardiness possible among all completions of R.

We get the following precedence among the jobs in N-R corresponding to various reverse sequences R. Note that as we are considering reverse sequences, if we have $i \leftarrow j$, then we fix j before i is fixed.

Node number	R	Precedence
4	(6 1)	$5 \leftarrow 7$
7	(6 7)	$3 \leftarrow 5$; $1 \leftarrow 5$
11	(3 6)	$7 \leftarrow 5$; $1 \leftarrow 5$
13	(3 6 5)	$1 \leftarrow 7$
17	(6 7 5)	$3 \leftarrow 1$
19	(3 6 5 7)	$2 \leftarrow 4$

Further, using the dominance theorem we see that the reverse sequence (6 3) is dominated by the reverse sequence (3 6) as the weighted tardiness for (6 3) is greater than that of (3 6). Thus we generate only 19 nodes out of the 8660 possible nodes to obtain the optimal solution. Elmaghraby (1968) has reported that 88 nodes were generated to obtain the optimal sequence using his method.

2.6 The $(n/1/F/T_u)$ problem with common due date

2.6.0 Introduction :

In this section we consider the single machine sequencing problem to minimise weighted tardiness when the due dates for the jobs are the same. The general procedures mentioned in the last section do not exploit the simplifications arising out of the jobs having a common due date. Lawler and Moore (1969) have considered a general functional equation which they have applied to this problem.

Section 2.6.1 deals with some results on optimal precedence among jobs. In section 2.6.2, we define a preferred sequence and note that it is sufficient to consider the set of all preferred sequences, while searching for an optimal sequence. A Branch and Bound algorithm is also given. While branching, the jobs are fixed in the reverse order, namely, the last

position in a sequence is fixed in the first stage and the last but first position is fixed next and so on. The bound U_C discussed in section 2.2 under special cases is applicable as the matrix C turns out to be monotone, with common due dates. Application of a lemma proved (lemma 2.6.6) reduces considerably the number of nodes to be created at any stage. Thus an algorithm incorporating these results is expected to do better than the general algorithms for the $(n/1/F/T_u)$ problem.

2.6.1 Optimal precedence among jobs

The problem considered is as in section 2.5, with $d_j = d$ for all j. Without loss of generality we assume $p_j > 0$ and $d < p(N)$ where $p(A) = \sum_{i \in A} p_i$ for any subset A of N. We follow the notations introduced in the section 2.5 regarding precedence among jobs.

We observe that in theorem 2.5.1 with $d_i = d$, Condition (i) is always satisfied for any two jobs j and k with $j < k$. Also, we have (x^*, y^*) given by lemma 2.5.1 as $(p', p(B_k))$ when $d < p' = p(A_j^c)$ since $d_k - d_j = 0 \leq p_k - p_j$. Thus, we have the following lemma.

Lemma 2.6.1 : For any two jobs j and k with $p_j \leq p_k$ if

$$u_j \geq u_k \quad \text{or} \quad 1 < u_k/u_j \leq \delta_{kj}$$

where $\delta_{kj} = \dfrac{p(A_j^c) - \text{Max}(p(B_k) + p_j, d)}{p(A_j^c) - \text{Max}(p(B_k) + p_k, d)}$

then $j \leftarrow k$.

Next we observe that theorem 2.5.2 is not applicable in this case as Condition (i) of the theorem is never satisfied. However, we prove the following lemma.

Lemma 2.6.2 : For any two jobs j and k with $p_j \leq p_k$ if

(i) $u_k \geq u_j$

and (ii) $\dfrac{p(A_k^c)-d}{p_j} \leq u_k/u_j$ then $k \leftarrow j$.

Proof : Let jobs j and k be such that $p_j \leq p_k$ and satisfy Conditions (i) and (ii) of the lemma. Let S be a sequence in which B_i appears before i for all i and job j appears before job k. Let S_j and C_k be the starting time of job j and completion time of job k respectively. Let S' be a sequence obtained from S by advancing all the jobs after j upto k by p_j and completing job j at C_k. It is sufficient to show that $\Delta_j + \Delta_k \leq 0$ where Δ_j and Δ_k are as defined in the proof of theorem 2.5.1.

Now,

$$\Delta_j = u_j[\text{Max}(C_k, d) - \text{Max}(S_j + p_j, d)]$$

and $\Delta_k = u_k[\mathrm{Max}(C_k - p_j, d] - \mathrm{Max}(C_k, d)]$.

Case 1. $C_k \leq d$. Then $\Delta_k = \Delta_j = 0$. So $\Delta_j + \Delta_k = 0$.

Case 2. $C_k > d$.

We have $C_k \leq p(A_k^c)$. Now $\Delta_j + \Delta_k \leq 0$ if

$$[C_k - \mathrm{Max}(S_j + p_j, d)]/[C_k - \mathrm{Max}(C_k - p_j, d)] \leq u_k/u_j.$$

Now either $C_k - p_j > d$ or $C_k - p_j \leq d$. First let $C_k - p_j \leq d$.

Then, $S_j + p_j \leq C_k - p_k \leq C_k - p_j \leq d$.

Therefore, $[C_k - \mathrm{Max}(d, S_j+p_j)]/[(C_k - \mathrm{Max}(d, C_k-p_j)]$

$$= 1 \leq u_k/u_j$$

by hypothesis (i).

Then, $\Delta_j + \Delta_k \leq 0$.

Next let $C_k - p_j > d$. Then

$$[C_k - \mathrm{Max}(d, S_j + p_j)]/[C_k - \mathrm{Max}(d, C_k - p_j)] \leq \frac{[C_k - d]}{p_j} \leq u_k/u_j$$

by hypothesis (ii) since $C_k \leq p(A_k^c)$. So $\Delta_j + \Delta_k \leq 0$.

Thus $\Delta_j + \Delta_k \leq 0$ in all cases and the lemma is proved.

Given any sequence, N can be partitioned into three sets π_1, π_2, π_3 as follows :

$$\pi_1 = \{i \mid i \in N,\ C_i \leq d\}$$

$$\pi_2 = \{i \mid i \in N,\ t_i < d \text{ and } C_i > d\}$$

and $$\pi_3 = \{i \mid i \in N,\ t_i \geq d\}$$

where t_i is the starting time of job i.

The following lemma gives a necessary condition for a sequence with $\pi_2 \neq \phi$ to be optimal.

Lemma 2.6.3 : A necessary condition for a sequence S with $\pi_2 = \{k\}$ to be optimal is

$$u_j/u_k \geq \begin{cases} 1 & \text{if } p_j \geq C_k - d \\ p_j/[C_k - d] & \text{if } p_j < C_k - d \end{cases}$$

for every $j \in \pi_1$.

Proof : Let S be an optimal sequence in which job $k \in \pi_2$ and $j \in \pi_1$. Now either $p_j \geq C_k - d$ or $p_j < C_k - d$.

Case 1. $p_j \geq C_k - d$.

Suppose $u_k > u_j$. Consider S' obtained from S by interchanging j and k. It is easy to see that S' has weighted

tardiness less than that of S, a contradiction. So $u_k \leq u_j$ for $j \in \pi_1$ and $p_j \geq C_k - d$, if S were to be optimal.

Case 2. $p_j < C_k - d$.

In this case, consider the sequence obtained from S by advancing all the jobs after j upto k by p_j and completing j at C_k, with the remaining jobs unaffected.

Now, we have

$$\Delta_j = u_j (C_k - d)$$

and $$\Delta_k = u_k[\text{Max}(C_k - p_j, d) - C_k]$$

$$= -u_k p_j \quad \text{since } C_k - d > p_j.$$

Therefore $\Delta_j + \Delta_k = u_j(C_k - d) - u_k p_j$.

Since S is optimal we have $\Delta_j + \Delta_k \geq 0$,

so $u_j/u_k \geq p_j/(C_k - d)$.

This completes the proof of the lemma.

The above lemma shows that an assertion made by Lawler and Moore (1969) regarding π_2 jobs is incorrect. They assert that the job k in π_2 must possess an u_k no greater than that of any job in π_1. Their assertion will be correct when there is no $j \in \pi_1$ such that $p_j < C_k - d$.

2.6.2 Reverse Sequences, Lower Bounds and Branching

In this section we will be dealing with ''reverse sequences''. Corresponding to a permutation schedule, we define a reverse sequence to be $R = [j_1, j_2, \ldots, j_q]$ where job j_1 is processed last, job j_2 is processed immediately before j_1 and so on. As in section 2.6.1 let t_j denote the starting time of job j. We call R a partial reverse sequence (PRS) if $q < n - 1$ and $t_{j_q} > d$. Otherwise R is called a complete reverse sequence (CRS). The motivation for this definition is that if R is a CRS then it determines π_3 and π_2 and thus the weighted tardiness.

Let $W(R)$ be the minimum weighted tardiness possible with the restriction imposed by R. If R is complete with $\pi_3 = [j_1 \ldots j_q]$, then $W(R)$ is given by

$$W(R) = \sum_{i=1}^{q} u_{j_i} T_{j_i} + (t_{j_q} - d)\, u_{j_{q+1}} \qquad \ldots 2.6.1$$

where T_{j_i} is the tardiness corresponding to job j_i.

The following lemma provides a lower bound for $W(R)$ when R is a PRS.

Lemma 2.6.4 : Let $R = [j_1 \ldots j_q]$ be any PRS and

$$L_1(R) = \sum_{s=1}^{q} u_{j_s}(C_{j_s} - d) + (t_{j_q} - d)\, u_R, \qquad \ldots 2.6.2$$

where $u_R = \underset{i \in R^c}{\text{Min}}\ u_i$.

Then $L_1(R) \leq W(R)$.

Proof : We recall that $q < n - 1$ and $t_{j_q} > d$ as R is a PRS. For any complete reverse sequence $R' = [j_1, \ldots, j_n]$ obtained by adding to R jobs in N-R in an optimal manner,

$$W(R) = \sum_{s=1}^{n} u_{j_s} T_{j_s} \geq \sum_{s=1}^{q+1} u_{j_s} T_{j_s} .$$

But $T_{j_{q+1}} = \text{Max}(C_{j_{q+1}} - d, 0)$

$= \text{Max}(t_{j_q} - d, 0) = t_{j_q} - d$ since

R is a PRS. By definition $u_R \leq u_{j_{q+1}}$ as $j_{q+1} \in$ N-R, and $C_{j_s} > d$ for $1 \leq s \leq q$ as R is a PRS. Hence

$$W(R) \geq \sum_{j_s \in R} u_{j_s} (C_{j_s} - d) + u_R (t_{j_q} - d)$$

$$= L_1(R).$$

This proves the lemma.

This lemma provides a good lower bound if there exists an $i \in$ N-R such that $p_i \geq t_{j_q} - d$. In case there is no such $i \in$ N-R, then we can improve the lower bound by using the lemma 2.2.4 specialized for the case of linear penalty rates with common due dates. Let $R = [j_1 \ldots j_q]$ be a PRS. Let $p_{n+1} = p_i$

Consider the $(r/1/F/T_u)$ *problem with common due dates.*

Let W^* be the weighted tardiness for an optimal sequence.

Let $(s_1 \ldots s_r)$ be a sequence of the r jobs such that

$$u_{s_1} \leq \ldots \leq u_{s_r}.$$

Let $(\ell_1 \ldots \ell_r)$ be a sequence of the r jobs such that

$$p_{\ell_1} \geq \ldots \geq p_{\ell_r}.$$

As $d_i = d$ for all i any sequence is an EDD sequence. So lemma 2.2.4 reduces to : An SPT sequence is optimal in case it is same as an LPR sequence. Moreover, the matrix C in proposition 2.2.1 is monotone as an EDD sequence coincides with an LPR sequence. Thus we have the lower bound corresponding to u_C defined in section 2.2.

Lemma 2.6.5 : Let $(s_1 \ldots s_r)$ and $(\ell_1 \ldots \ell_r)$ be as defined above and $y_j = \text{Max}[\sum_{i=j}^{r} p_{\ell_i} - d, 0]$.

Then $\sum_{j=1}^{r} y_j u_{s_j} \leq W^*$. Further, if $s_i = \ell_i$ for all i, then $(s_1 \ldots s_r)$ gives an optimal reverse sequence and

$$\sum_{j=1}^{r} y_j u_{s_j} = W^*.$$

In the light of this lemma, we get a better lower bound L(R) as given below.

$$W(R) \geq L(R) = \sum_{i=1}^{q} (C_{j_i} - d)\, u_{j_i} + C(R) \qquad \ldots 2.6.3$$

where $C(R) = \sum_{k=1}^{n-q} u_{s_k} y_k$ where u_{s_k}, y_k are as defined above corresponding to the jobs in N-R. L(R) given by 2.6.3 coincides with $L_1(R)$ given by 2.6.2 in case there exists an $i \in N-R$ such that $p_i \geq t_{j_q} - d$. Otherwise, $L(R) > L_1(R)$ for any R, a PRS.

From now on we assume the jobs are numbered such that $u_1/p_1 \leq \ldots \leq u_n/p_n$ i.e. $(n, n-1, \ldots, 1)$ is a Mc Naughton's sequence.

Definition 2.6.1 : Let $R = (j_1, j_2, \ldots, j_k)$ be any CRS. Let the corresponding π_3 be $\{j_1 \ldots j_q\}$. Then R is said to be a preferred reverse sequence if either $q \leq 1$ or $q \geq 2$ and

$$j_1 < j_2 < \ldots < j_q.$$

The way jobs are numbered, it can be easily seen that any CRS can be converted to a preferred reverse sequence by rearranging jobs in π_3, without increasing the weighted tardiness. Thus we may confine our search for an optimal CRS, only to the set of all preferred reverse sequences, denoted by $\underline{E}$.

Lemma 2.6.6 : Let $R = [j_1, j_2 \ldots j_q]$ be a PRS. While searching for an optimal completion of R it is sufficient

to consider jobs in $A(\_)B$ for the $q+1^{th}$ position, where

$$A = \{i \mid i \in N\text{-}R,\ i > j_q,\ p_i < t_{j_q} - d\}$$

$$B = \begin{cases} D & \text{if } F = \phi \\ \phi & \text{if } F \neq \phi \end{cases}$$

with

$$F = \{i \mid i \in D,\ i < j_q,\ p_i = t_{j_q} - d\}$$

$$D = \{i \mid i \in C,\ u_i = \underset{j \in C}{\text{Min}}\ u_j\}$$

and $C = \{i \mid i \in N\text{-}R,\ p_i \geq t_{j_q} - d\}$.

Further, when $j_{q+1} \in A$, the resulting reverse sequence is a PRS ; when $j_{q+1} \in B$ it is a CRS. If $B \neq \phi$, it is sufficient to consider any one $i \in B$ as j_{q+1}.

Proof : It is easy to see that when $j_{q+1} \in A$ the resulting reverse sequence $[j_1, \ldots, j_{q+1}]$ is a PRS since for $i \in A$, $p_i < t_{j_q} - d$. Similarly when $j_{q+1} \in D$, the resulting reverse sequence is a CRS and for all $j_{q+1} \in D$, the weighted tardiness of $[j_1, j_2, \ldots, j_q, j_{q+1}]$ is the same because $u_{j_{q+1}}$ and the tardiness for j_{q+1} are constant for all choices of j_{q+1} from D. So it is sufficient to consider any one $i \in D$ as j_{q+1}.

Thus, it remains to show that it is sufficient to consider $j_{q+1} \in A(\_)B$.

Suppose $j \notin A$. Then either (i) $j \in C$ or (ii) $p_j < t_{j_q} - d$ and $i < j_q$.

First, let j be such that (ii) is satisfied. Clearly, then $Rj = [j_1, j_2, \ldots, j_q, j]$ will not give rise to a preferred sequence. Hence Rj need not be considered. Next let $j \in C$. Then the tardiness is the same for any $j_{q+1} \in C$ and $W(Rj) \geq W(Ri)$ for any $i \in D$, and so it is sufficient to consider jobs in D for $q+1^{th}$ position.

So let $j \in D$. Now if $F \neq \phi$, then there exists an $i \in D$ such that $i < j_q$ and $p_i = t_{j_q} - d$. This implies that Ri is not a preferred reverse sequence. Let R' be a rearrangement of Ri such that R' is a preferred reverse sequence.

We know that $W(Ri) \geq W(R')$ and $W(Rj) = W(Ri)$ as $i, j \in D$. So when $j \in D$ and $F \neq \phi$, Rj need not be considered. Thus, when $j \notin A$, j is to be considered only when $j \in D$ and $F = \phi$ i.e. when $j \in B$.

Hence it is sufficient to consider $j \in A \cap B$, for the $q+1^{th}$ position in any completion of R.

This completes the proof.

Example 2.6.1 : Consider the $(6/1/F/T_u)$ problem given below with $d = 10$.

i	1	2	3	4	5	6
p_i	5	3	4	4	6	1
u_i	1	1	3	4	6	2

We have $p(N) = 23$. Consider $R = (3,5)$. Since $t_{j_2} = t_5 = 13$ and $N-R = \{1,2,4,6\}$, we have

$A = \{6\}$ since $6 > 5$, $p_6 = 1 < 13 - 10 = 3$

$C = \{1,2,4\}$ since $p_i \geq 3$, $i = 1,2,4$

$D = \{1,2\}$ as $u_1 = u_2 = 1 = \underset{j \in C}{\text{Min}}\, u_j$

$F = \{2\}$ as $p_2 = 3$

Thus $B = \phi$ as $F \neq \phi$.

So we need to consider only [3,5,6] as the descendant of R. It is easy to verify, on the other hand that for $R = (2,6)$ both A and B are empty.

An algorithm

A Branch and Bound algorithm incorporating the results obtained in section 2.6.1 and in this section, is given below. The nodes of the tree represent the PRS's. Let $[\phi]$ represent the root of the tree corresponding to the empty reverse sequence. Initially let $H = \{[\phi]\}$.

Step 0 : Use the precedence lemmas and get the corresponding precedence matrix. If all the jobs are ordered, let R* be the CRS thus obtained with W* as weighted tardiness and go to Step 7. Otherwise, discard all the jobs that are shown to be in some definite positions. Renumber the remaining jobs as in reverse Mc Naughton's sequence. Let n be the number of jobs remaining. Go to Step 1.

Step 1 : Take $R^* = (1,2,\ldots,n)$ and calculate $W^* = W(R^*)$ using 2.6.1. Choose the node $[\phi]$ for branching and go to Step 2.

Step 2 : Let $R = [j_1, j_2, \ldots, j_q]$ be the node chosen for branching. Drop R from H. Find sets A and B as given in lemma 2.6.6. If $A \cap B = \phi$, then go to Step 5. Otherwise, branch with $Ri = (j_1 \ldots j_q\ i)$ for each $i \in A \cap B$ such that there does not exist any $j \in N-R$ with $i \leftarrow j$ (with the restriction that R is fixed). Include such Ri in H and go to Step 3.

Step 3 : If Ri is partial, i.e. $i \in A$, calculate

$$L(Ri) = C(Ri) + (t_{j_q} - d)\, u_i + L'(R)$$

where $L'(R) = \sum_{s=1}^{q} T_{j_s}\ u_{j_s}$ and $C(Ri)$

is as given in lemma 2.6.5, with the jobs in the set N-Ri. Set $q = q+1$. Store $[L(Ri), L'(Ri), t_{j_q} - d]$ for each PRS along with q. Go to Step 4.

If Ri is complete i.e. $i \in B$, then calculate for any one $i \in B$,

$$W(Ri) = L'(R) + (t_{j_q} - d)\, u_i$$

and go to Step 6.

Step 4 : Discard from H all R such that $L(R) \geq W^*$. Go to Step 5.

Step 5 : Find R such that L(R) is the least for $R \in H$. In case of ties choose an R with maximum q. Go to Step 2. If $H = \phi$, then go to Step 7.

Step 6 : If $W(Ri) < W^*$ for $i \in B$ then store

$$W^* = W(Ri)$$

$$R^* = Ri.$$ Go to Step 4.

Otherwise discard Ri, $i \in B$ from further consideration. Go to Step 5.

Step 7 : An optimal CRS has been reached. W^* is the optimal weighted tardiness and R^* is the corresponding CRS. Stop.

2.7 Single Machine sequencing, with Intermittent Job Arrivals, to Minimise the Number of Late Jobs

2.7.0 Introduction :

We consider the (n/1/F/f) problem with f, number of late jobs, under the usual assumptions but for assumption A.2. In all earlier sections the jobs were assumed to be available at the same time (Assumption A.2). This assumption is referred to as simultaneity of job arrivals. If the jobs are available for processing at different time points, then we say, it is a case of intermittent job arrivals.

If preemption is allowed at the time a new job arrives, there are two priority rules available, namely preemptive repeat and preemptive resume. In the non-preemptive case a job taken up for processing on a machine has to be completed before another job can be taken up. Here, we consider the problem in which the job arrivals are known apriori and there is a common due date; our objective is to minimise the number of late jobs. The problem is solved by showing that it corresponds to the problem of sequencing through a single machine to minimise the number of late jobs, with different due dates for jobs which are available for processing at the same time. Moore (1968) gives an algorithm for solving the later problem. An Example is worked out.

2.7.1 Main Theorem and Example

There are n jobs to be processed through a machine. But the jobs arrive at different times, known apriori. There is a common due date, for all the jobs. The problem considered is that of finding a sequence that minimises the number of late jobs.

Let r_j be the time at which job j arrives. Let d be the common due date.

Let $p = (i_1\ i_2 \ldots i_n)$ be a sequence of jobs in N.

Then,

$$C_{i_1} = r_{i_1} + p_{i_1}$$

$$C_{i_2} = \text{Max}(r_{i_1} + p_{i_1},\ r_{i_2}) + p_{i_2}$$

$$\vdots$$

$$C_{i_n} = \text{Max}\left(r_{i_1} + \sum_{j=1}^{n-1} p_{i_j},\ r_{i_2} + \sum_{j=2}^{n-1} p_{i_j}, \ldots, r_{i_n}\right) + p_{i_n}$$

Let $K_{u,v} = r_{i_u} + \sum_{j=u}^{v} p_{i_j}$

Now

$$C_{i_v} = \max_{1 \le u \le v} [K_{u,v}]$$

We will call the above problem as Problem 1 and consider another problem called Problem 2 given as follows :

Let $d_i = d - r_i$

Let d_i be the due date for job i, $i=1,\ldots,n$, and let the jobs be available at time zero, for processing. The problem is to maximise the number of early jobs. To solve problem 1, we use the following theorem, which relates the solutions of problem 1 and problem 2.

Theorem 2.7.1 : *For any optimal sequence S for problem 1* there exists an optimal sequence S' for problem 2, in which, the early jobs of S are early and appear in the reverse order and vice versa.

Proof : Let S_E and S_L be the ordered set of early jobs and the ordered set of late jobs in S, respectively. Let e = number of elements in S_E and let C_i be the completion time of job i in S.

Let $S_E = (i_1, i_2 \ldots i_e)$.

Now, define S' as follows :

$$S' = (i_e\ i_{e-1} \ldots i_1\ ;\ S_L).$$

Let C'_i be the completion time of job i in S' for the problem 2.

In S, we have, $C_{i_{j+1}} \geq C_{i_j} + p_{i_{j+1}}$, for all j.

So $d - C_{i_j} \geq d - C_{i_{j+1}} + p_{i_{j+1}}$, for all j.

Now consider, S',

$$C'_{i_j} = \sum_{s=j}^{e} p_{i_s}, \quad 1 \leq j \leq e$$

and $C'_{i_e} = p_{i_e} \leq d - C_{i_e} + p_{i_e}$ since i_e is early in S

$$C'_{i_{e-1}} = p_{i_e} + p_{i_{e-1}} \leq (d - C_{i_e} + p_{i_e}) + p_{i_{e-1}}$$

$$\leq d - C_{i_{e-1}} + p_{i_{e-1}}$$

and so on.

Thus, $C'_{i_j} \leq d - C_{i_j} + p_{i_j}, \quad 1 \leq j \leq e.$

We require, $C'_{i_j} \leq d_{i_j}, \quad 1 \leq j \leq e$

Now

$$C'_{i_j} \leq d - C_{i_j} + p_{i_j}$$

$$= d - \max_{1 \leq u \leq j} K_{u,j} + p_{i_j}$$

$$\leq d - K_{j,j} + p_{i_j}$$

$$= d - (r_{i_j} + p_{i_j}) + p_{i_j}$$

$$= d - r_{i_j} = d_{i_j} \qquad 1 \leq j \leq e$$

Thus every job which is early in S is early in S' also. From the way S' is constructed it is clear that

$S' = (R(S_E), S_L)$ where $R(S_E)$ is the reverse sequence of S_E. Conversely if S' is any optimal sequence for problem 2 and S is obtained from S' by reversing the early jobs of S', it can be similarly proved that any job which is early in S' is early in S also. Thus in fact the sets of early jobs in S and S' are the same. This completes the proof of the theorem.

From the above discussion it is clear that, solving problem 1 is equivalent to solving problem 2. Problem 2 can be solved easily using Moore's Rule (1968). We give below Moore's Rule to find an optimal sequence for the (n/1/F/f) problem with f as the number of late jobs, under usual assumptions.

Step 1 : Arrange the jobs according to the non-decreasing order of d_j's,

i.e. $d_{i_1} \leq d_{i_2} \leq \ldots \leq d_{i_n}$

Set $k = 1$. Go to Step 2.

Step 2 : Find a positive integer $s \leq n$ such that

$$\sum_{j=1}^{u} p_{i_j} \leq d_{i_u} \quad \forall \quad i \leq u \leq s-1$$

and

$$\sum_{j=1}^{s} p_{i_j} > d_{i_s}.$$

Make job i_q a late job where $p_{i_q} = \underset{1 \le j \le s}{\text{Max}} \; p_{i_j}$. Remove that from further consideration. Consider the reduced problem and go to Step 1 with $n = n - 1$ and $k = k + 1$.

If no such s exists, i.e.,

$$\sum_{j=1}^{s} p_{i_j} \le d_{i_s} \quad \forall \quad s \le n$$

then stop. We have got an optimal solution with the number of late jobs equal to $k - 1$.

We have the following proposition from Moore's Rule.

Proposition 2.7.1 : There exists an optimal sequence for Problem 1 in which the early jobs are processed in the non-decreasing order of r_j's.

We illustrate the method with the data given below.

Example 2.7.1

d = 15

j	r_j	p_j	d_j
1	12	2	3
2	10	6	5
3	8	4	7
4	5	3	10
5	2	4	13
6	1	3	14

The jobs are already numbered according to the non-decreasing order of d_j's.

We have, $k = 1$

$2 \leq 3$

$2 + 6 > 5$. So $s = 2$ and job 2 is late.

$k = 2$

$2 \leq 3$

$2 + 4 \leq 7$

$2 + 4 + 3 \leq 10$

$2 + 4 + 3 + 4 \leq 13$

$2 + 4 + 3 + 4 + 3 > 14$. So $s = 5$ and job 5 is late.

$k = 3$. Then s does not exist, so the optimal number of late jobs is 2 and the early jobs are (1,3,4,6) for problem 2.

Therefore, (6,4,3,1,2,5) is an optimal sequence for problem 1.

CHAPTER 3

GROUPING PROBLEMS

3.0 Introduction :

In this chapter we consider the parallel sequencing problem, the batch splitting problem and the cluster analysis problem. The former two are scheduling problems, while the latter has a wider application outside scheduling context.

The parallel sequencing problem is the (n/(m)/F/f) problem, with a single stage having m parallel machines for processing. Each job has to be processed only once and this can be done on any one of these m machines. The problem is to find a schedule which is equivalent to a partition of the set of jobs, N, into m ordered subsets so as to minimize f. This problem with f as weighted sum of completion times has been considered by Mc Naughton (1959) and he has also given a simple rule to get the optimal sequence when m=1 and f is the weighted sum of completion times. Eastman and others (1964) have derived lower and upper bounds on the optimal weighted sum of completion times in the m identical machines case. Lawler (1964) has considered the dynamic programming approach for the (n/1/F/u(C)) problem and suggests that it can be extended to m parallel machines case as well. Lawler has also given a transportation problem equivalent to the m parallel machines problem when the

processing times for all the jobs are same, with $u(C) = \sum_{j=1}^{n} u_j(C_j)$ and u_j's are nondecreasing functions of C_j. When processing times are different he gives an equivalent restricted transportation problem.

Root (1965) has considered the $(n/(m)/F/\bar{T})$ problem with common due date and identical machines. Rothkopf (1966) has applied the dynamic programming approach to the general m machine problem. Gupta and Walvekar (1969) have formulated this problem as a 0-1 mixed integer programming problem. Arthanari and Ramamurthy (1970) have given a Branch and Bound algorithm for the $(n/(m)/F/C_u)$ problem. Nabashima (1971) has described a method for solving parallel sequencing problems using his general algorithm based on the disjunctive graph formulation of the problem.

In section 3.1 we discuss the $(n/(m)/F/C_u)$ problem and give a few lower bounds on the weighted sum of completion times of schedules with the restriction that certain subset of jobs is to be processed on some specific machines in a given order. An algorithm given in Arthanari and Ramamurthy (1970) is stated incorporating some of these bounds. When the machines are identical further simplifications are achieved in the algorithm.

In section 3.2 we consider the $(n/m/R/F_{max})$ problem and give a disjunctive graph formulation of the problem. When the jobs are identical the problem of grouping them into m batches and processing them through the machines so as to minimize the total elapsed time under the restriction that the batches once formed are not allowed to be split further or grouped together, is called the batch splitting problem. A simple rule for *finding an optimal solution to the problem is given when* n *is* divisible by m or n is sufficiently large.

Section 3.3 considers the well known cluster analysis problem. The problem of cluster analysis can be stated as follows : A set of n items is to be partitioned into m nonempty subsets in such a manner that the combined within groups sums of squares is minimised. This problem has been considered by Rao (1952), Ward (1963), Edward and Cavalli-Sforza (1965), Jenson (1969), Vinod (1969) and Ruspini (1969) to mention a few. An extensive bibliography of the work done is found in Jenson (1969). Such problems are encountered in Taxonomy, Sociology, Anthropometry and Industry.

An integer programming formulation of the problem is given by Vinod (1969). A dynamic programming algorithm for clustering is given by Jenson (1969). These procedures assure an optimal solution while there are other partial search

procedures which do not assure optimality, for example, Edward and Cavalli-Sforza (1965). An implicit enumeration method is given for this problem in section 3.3 applying the algorithm given in section 3.1 with bounds obtained by solving certain less restricted problems. An example is given to illustrate the algorithm.

3.1 Parallel Sequencing Problem

3.1.1 Main Results on the $(n/(m)/F/C_u)$ problem

There are n jobs to be processed and m machines are available for processing. Each job has to be processed only once and this can be done on any of the m machines. The processing time required for job j on machine i is p_{ij} (assumed to be greater than zero) and u_j is the penalty rate (assumed to be greater than zero) for job j.

Let $A \subseteq N$ and ℓ be the number of jobs in A. Let $W_i(A)$ be the optimal weighted sum of completion times for the $(\ell/1/F/C_u)$ problem where the jobs are those in A and the machine considered is the i^{th} machine. We know that $W_i(A)$ is attained for a Mc Naughton's sequence of the jobs in A, that is,

$$W_i(A) = \sum_{v=1}^{\ell} \sum_{s=1}^{v} u_{j_v} p_{ij_s} \qquad \ldots 3.1.1$$

when $A \neq \phi$

where

$$p_{ij_1}/u_{j_1} \leq \ldots \leq p_{ij_\ell}/u_{j_\ell}$$

If $A = \phi$ we take $W_i(A) = 0$.

In this section we use the following notation.

$$p_j = \min_{1 \leq i \leq m} p_{ij}, \quad p^*(A) = \min_{\substack{1 \leq i \leq m \\ j \in A}} p_{ij},$$

and $$u^*(A) = \min_{j \in A} u_j.$$

Lemma 3.1.1 : For any two disjoint subsets A_1 and A_2 of N, we have

$$W_i(A_1 \cup A_2) \geq W_i(A_1) + W_i(A_2) \quad \text{for all } i = 1,2\ldots m.$$

Proof : Let q_1 and q_2 be the number of jobs in A_1 and A_2 respectively. Consider $\sigma = (j_1 \ldots j_{q_1+q_2})$, a Mc Naughton sequence corresponding to the jobs in $A_1 \cup A_2$ and the i^{th} machine. Let $\sigma_1 = (j_{11},\ldots,j_{1q_1})$ & $\sigma_2 = (j_{21},\ldots,j_{2q_2})$ be the order of jobs in A_1 and A_2 as they appear in σ, respectively. It is easy to see that σ_1 and σ_2 are Mc Naughton sequences corresponding to jobs in A_1 and A_2 respectively. Consider any $j_k \in A_1 \cup A_2$. It belongs to one and only one of the sets A_1 and A_2. Suppose $j_k = j_{1s} \in A_1$.

Then, $\{j_1,\ldots,j_k\} \supseteq \{j_{11},\ldots,j_{1s}\}$.

Hence, the job j_k is completed in σ later than in σ_1 and so the weighted completion time of the job in σ is greater than or equal to that of the job in σ_1, Similarly for a job in A_2 this holds. The lemma follows immediately.

Lemma 3.1.2 : Let $A_1,A_2,\ldots,A_m$ be any m disjoint subsets of N, $A = \bigcup_{i=1}^{m} A_i$ and let the number of jobs in A be ℓ. Then we have,

$$\sum_{i=1}^{m} W_i(A_i) \geq p^*(A).\ u^*(A).\ (k+1)\ [r + \frac{m.k}{2}]$$

where $k = [\ \ell\ /\ m\ ]$ and $r = \ell - mk$.

Proof : Let q_i be the number of jobs in A_i.

Then

$$W_i(A_i) = \sum_{v=1}^{q_i} \sum_{s=1}^{v} u_{j_v}\ p_{ij_s}$$

$$\geq u^*(A).\ p^*(A)\ [\frac{q_i\ (q_i+1)}{2}]$$

by substituting $P^*(A)$ instead of p_{ij_s} for all i and s and $u^*(A)$ instead of u_{j_s} for all s.

So

$$\sum_{i=1}^{m} W_i(A_i) \geq u^*(A).p^*(A)\ [\ \sum_{i=1}^{m} \frac{q_i\ (q_i+1)}{2}]$$

But the minimum of $\sum_{i=1}^{m} \frac{q_i(q_i + 1)}{2}$

subject to $\sum_{i=1}^{m} q_i = \lambda$ *with* q_i *nonnegative integers, is given by* $(k+1)\,[r + \frac{m.k}{2}]$ with k and r as defined in the lemma.

This proves the lemma.

In what follows, by a partition of a set $A \subseteq N$ we mean an ordered partition of A into m subsets.

Definition 3.1.1 :

Let $B = \{B_i,\ i = 1,2,\ldots,m\}$ be any partition of a subset of job in N. Any partition $D = \{D_i,\ i = 1,2,\ldots,m\}$ of the set of all the n jobs is said to be a completion of B in case $B_i \subseteq D_i$ for every i. By the weighted sum of completion times of the partition D, we mean $\sum_{i=1}^{m} W_i(D_i)$.

Definition 3.1.2 :

Let $\overline{W}(B)$ be the minimum possible weighted sum of completion times for any completion of B.

Theorem 3.1.1 : Let $B = \{B_i,\ i = 1,2,\ldots,m\}$ be a partition of a subset, N_q, of q jobs. Let $\overline{N}_q = N - N_q$. Then we have,

$$\overline{W}(B) \geq \sum_{i=1}^{m} W_i(B_i) + p^*(\overline{N}_q)\ .\ u^*(\overline{N}_q)((k+1)(r + \tfrac{mk}{2}))$$

where $k = [(n-q)/m]$ and $r = n-q-mk$.

Proof : Let $D = \{D_i, i = 1,2,\ldots,m\}$ be a completion of B. Then $B_i \subseteq D_i$ for all i. Let $G_i = D_i - B_i$. It is easily seen that $\bigcup_{i=1}^{m} G_i = \bar{N}_q$ and the number of jobs in $\bar{N}_q$ is $n-q$.

Now the weighted sum of completion times for D is

$$\sum_{i=1}^{m} W_i (D_i) = \sum_{i=1}^{m} W_i (B_i \cup G_i)$$

$$\geq \sum_{i=1}^{m} W_i (B_i) + \sum_{i=1}^{m} W_i (G_i)$$

by lemma 3.1.1

$$\geq \sum_{i=1}^{m} W_i (B_i) + p^*(\bar{N}_q)\, u^*(\bar{N}_q)\left\{(k+1)\left(r + \frac{mk}{2}\right)\right\}$$

by lemma 3.1.2

The proof is complete since D is any arbitrary completion of B.

This theorem provides a lower bound for $\bar{W}(B)$, which is used in a Branch and Bound algorithm developed in Arthanari and Ramamurthy (1970). This algorithm incorporating some lower bounds obtained in section 3.1.2 is given there.

Since an optimal order of processing any subset of jobs in a machine is given by a Mc Naughton's sequence, it is enough to specify the machine where a job is to be processed. Such

a specification is called an assignment and $\sigma = (i_1, i_2, \ldots, i_q)$ denotes an assignment of the first q jobs, where i_v denotes the machine to which job v is allotted. We call σ partial or complete according as $q < n$ or $q = n$ and q is called the order of that assignment. Thus with a partition B we associate a σ by specifying that the jobs in the i^{th} set of B will be processed on machine i and vice versa.

3.1.2 Some More Lower Bounds on $\overline{W}(B)$ and an algorithm

Eastman and others (1964) have given a lower bound for the weighted sum of completion times when the machines are identical. Using that bound for the jobs in $\overline{N}_q$ we can get the following lower bound for $\overline{W}(B)$:

$$\overline{W}(B) \geq \sum_{i=1}^{m} W_i (B_i) + E(\overline{N}_q)$$

where

$$E(\overline{N}_q) = \frac{n-q+m}{m(n-q+1)} \sum_{v=1}^{n-q} u_{j_v} \sum_{s=1}^{v} p_{j_s}$$

and $(j_1, \ldots, j_{n-q})$ is a Mc Naughton's sequence of jobs in $\overline{N}_q$.

Next, we note that the lower bound given by theorem 3.1.1 can be improved easily by observing the fact that the completion time of a job is a sum of its waiting time and its processing time.

We have,

$$\overline{W}(B) \geq \sum_{i=1}^{m} W_i(B_i) + \beta(\overline{N}_q)$$

where

$$\beta(\overline{N}_q) = \sum_{j \varepsilon \overline{N}_q} u_j p_j + p^*(\overline{N}_q)\, u^*(\overline{N}_q) h$$

and $h = k[n - q - \frac{m(k+1)}{2}]$ with $k = [\frac{n-q}{m}]$ ... 3.1.2

When we have a partial assignment with each of the machines having at least one job we have the following result :

Lemma 3.1.3 : Let $B = \{B_i, i = 1,2,\ldots,m\}$ be any partition of a subset N_q of q jobs with $B_i \neq \phi$ for all i. Let $\overline{N}_q = N - N_q$. Thenwe have

$$\overline{W}(B) \geq \sum_{i=1}^{m} W_i(B_i) + \upsilon(\overline{N}_q)$$

where

$$\upsilon(\overline{N}_q) = \sum_{j \varepsilon \overline{N}_q} S_j + \sum_{j \varepsilon \overline{N}_q} u_j p_j$$

$$S_j = \min_{1 \leq i \leq m} [u_j \sum_{s \leftarrow j} p_{is} + p_{ij} \sum_{j \leftarrow s} u_s]$$

where the first summation is over jobs that precede j and the second summation is over jobs that succeed j in a Mc Naughton's sequence of the jobs in $B_i \cup \{j\}$.

Proof : Any job $j \varepsilon \overline{N}_q$ can be assigned to any one of the m machines. Let $W'(B_i \cup \{j\})$ be the increment in the

weighted sum of waiting times when job j is assigned to the i^{th} machine. Then clearly

$$W'(B_i \leftarrow \{j\}) = u_j \sum_{s<-j} p_{is} + p_{ij} \sum_{j<-s} u_s \quad \ldots 3.1.3$$

So,

$$S_j = \min_{1\leq i\leq m} [W'(B_i \leftarrow \{j\})]$$

gives the minimum possible increase when j is assigned.

Thus, $\sum_{j\epsilon\bar{N}_q} S_j$ gives the minimum possible increase in the weighted sum of waiting times when all the jobs are assigned, with the restriction imposed by B.

The result follows immediately by adding $\sum_{j\epsilon\bar{N}_q} u_j p_j$ and $\sum_{j\epsilon\bar{N}_q} S_j$ to $\sum_{i=1}^{m} W_i(B_i)$.

Note that $\upsilon(\bar{N}_q)$ given by the above lemma takes into account the actual assignment of jobs in N_q while $\beta(\bar{N}_q)$ and $E(\bar{N}_q)$ depend only on the set N_q.

We now consider two special cases of the $(n/(m)/F/C_u)$ problem and develop another lower bound using the optimal solutions of these cases. Let $A \subseteq N$ and ℓ be the number of jobs in A.

Case i $u_j = u^*(A)$ for all $j \in A$ and $p_{ij} = p_j$ for all i and j.

Let $(j_1, \ldots, j_\lambda)$ be an ordering of the jobs in A such that $p_{j_1} \leq p_{j_2} \leq \ldots \leq p_{j_\lambda}$.

In this case an optimal solution to the problem with the jobs in A is obtained by simply assigning jobs to the machines in rotation as follows :

job	j_1	j_2	...	j_m	j_{m+1}	j_{m+2}	...	j_{2m}	....	j_{km+1}	...	j_{km+r}
Machine	1	2	...	m	1	2	...	m	....	1	...	r

where $k = [\lambda / m]$ and $r = \lambda - mk$.

This result is immediate from Theorem 5 proved in Root (1965). This is also given in Maxwell and others (1967).

Here we have the optimal weighted sum of waiting times given by

$$\delta_1(A) = u^*(A) \left\{ k \sum_{s=1}^{r} p_{j_s} + \sum_{h=1}^{k-1} (k-h) \left(\sum_{s=1}^{m} p_{j_{(h-1)m+r+s}} \right) \right\} \quad \ldots 3.1.4$$

Case ii $p_{ij} = p^*(A)$ for all i and for all j.

Let $(j_1, j_2, \ldots, j_\lambda)$ be an ordering of the jobs in A

such that, $u_{j_1} \geq u_{j_2} \geq \ldots \geq u_{j_\lambda}$.

In this case an optimal solution to the problem is obtained, by assigning jobs to the machines in rotation as in Case i.

The proof of optimality of this solution can be given on the same lines as in Case i. We have the optimal weighted sum of waiting times given by

$$\delta_2(A) = p^*(A) \{ \sum_{h=1}^{k} (h-1)(\sum_{i=1}^{m} u_{j_{(h-1)m+i}}) + k \sum_{i=1}^{r} u_{j_{km+i}} \} \quad \ldots 3.1.5$$

Now consider a partition B of a subset N_q of q jobs. We have

$$\bar{W}(B) \geq \sum_{i=1}^{m} W_i(B_i) + \delta(\bar{N}_q)$$

where $\delta(\bar{N}_q) = \text{Max}[\delta_1(\bar{N}_q), \delta_2(\bar{N}_q)] + \sum_{j \epsilon \bar{N}_q} p_j u_j \quad \ldots 3.1.6$

with $\delta_1(\bar{N}_q)$ and $\delta_2(\bar{N}_q)$ given by 3.1.4 and 3.1.5 respectively.

The above inequality follows from the fact that for any partition of the jobs in $\bar{N}_q$, the weighted sum of waiting times decreases when the processing times p_{ij} or the penalty rates u_j are decreased.

Remarks :

1. $\beta(\bar{N}_q) \leq \delta(\bar{N}_q)$ since $\beta(\bar{N}_q) = \sum_{j \epsilon \bar{N}_q} p_j u_j$ is obtained by replacing all u_j's by $u^*(\bar{N}_q)$ in $\delta_2(\bar{N}_q)$.

2. $\delta(\bar{N}_q)$ also does not depend on the *actual assignment* of jobs to machines given by B.

3. In 60 problems considered for comparing $\delta(\bar{N}_q)$ with $E(\bar{N}_q)$, we find that in about forty cases $\delta(\bar{N}_q)$ dominates $E(\bar{N}_q)$.

An algorithm

The set of all assignments forms the set of nodes of a tree with root at the empty assignment denoted by $(\cdot)$. A pair of nodes, one of q^{th} order and the other of $(q+1)^{th}$ order, are connected by an arc when the assignment of the first q jobs in the $(q+1)^{th}$ order node is identical with that of the q^{th} order node and the $(q+1)^{th}$ order node is called a descendant of the q^{th} order node. Let σ denote any node $(i_1,\ldots,i_q)$ or $(\cdot)$ selected for branching. Let $\sigma_1, \sigma_2, \ldots, \sigma_m$ be the m descendants of σ obtained by assigning job $q+1$ to machines $1,2,\ldots,m$ respectively. Let $B = [B_1,\ldots,B_m]$ be the partition corresponding to σ. Let $B^k = [B_1^k, \ldots, B_m^k]$ be the partition

corresponding to σ_k, i.e.,

$$B_i^k = \begin{cases} B_i & \text{for all } i \neq k \\ B_i \left(\_\right) \{q+1\}, & i = k. \end{cases}$$

Let

$$\alpha(\sigma_k) = \sum_{\substack{i=1 \\ i \neq k}}^{m} W_i(B_i) + W_k(B_k^k) + \text{Max}[\upsilon(\bar{N}_{q+1}), \delta(\bar{N}_{q+1})] \qquad \ldots 3.1.7$$

where $\upsilon(\bar{N}_{q+1})$ is given by lemma 3.1.3 and $\delta(\bar{N}_{q+1})$ is given by 3.1.6.

Let $L((\cdot)) = \alpha((\cdot))$ initially, and let

$$L(\sigma_k) = \text{Max}[L(\sigma), \alpha(\sigma_k)] \qquad \ldots 3.1.8$$

$\delta(\bar{N}_q)$ can be tabulated in the beginning and used at any stage of computation. $\upsilon(\bar{N}_q)$ needs to be calculated at each node.

Step 0 : Find $\delta(\bar{N}_q)$ for $q = 0,1,2,\ldots,n-1$. Set $L((\cdot)) = \alpha((\cdot)) = \delta(N)$, $\pi = \{(\cdot)\}$ and $Z_o = \infty$ or the weighted sum of completion times of any complete assignment σ_o. Choose $(\cdot)$ for branching and go to Step 1.

Step 1 : Let σ be the q^{th} order node selected for branching. Delete this node from π. Generate its descendants $\sigma_1,\ldots,\sigma_m$ and go to Step 2.

Step 2 : *Calculate* $L(\sigma_k)$ *using* 3.1.8 *and go to* Step 3.

Note that

$$W_k(B_k^k) = W_k(B_k) + [u_{q+1} \sum_{s<-j} p_{ks} + p_{iq+1} \sum_{j<-s} u_s]$$

where the first summation is over jobs that precede j and the second summation is over jobs that succeed j in a Mc Naughton's sequence of the jobs in B_k $(\_)$ $\{q+1\}$.

Step 3 : Find $S = \{\sigma_k | L(\sigma_k) < Z_o\}$.

If $S = \phi$ then go to Step 7. Otherwise go to Step 4.

Step 4 : Find σ^* such that $L(\sigma^*) = \underset{\sigma_k \in S}{\text{Min}} L(\sigma_k)$.

In case of ties choose one with least, actual weighted sum of completion times for jobs in σ^*. If the order of σ^* is $< n$ then go to Step 5. Otherwise, go to Step 6.

Step 5 : Set $\pi = \pi$ $(\_)$ S. Choose σ^* for branching and go to Step 1.

Step 6 : Set $Z_o = L(\sigma^*)$

$\sigma_o = \sigma^*$ and delete all nodes σ with $L(\sigma) \geq Z_o$ from π and go to Step 7.

Step 7 : If $\pi = \phi$ then go to Step 8. Otherwise, choose a node σ of highest order from π for branching

and go to Step 1. In case of ties choose one such node σ with the smallest value of $L(\sigma)$.

Step 8 : σ_o is optimal with weighted sum of completion times, Z_o. In each machine the jobs are processed according to Mc Naughton's rule. Stop.

Remarks

1. When the order of the node σ, chosen for branching is n-1, we need to generate only one descendant σ_h of σ where h is such that $S_n = W'(B_h\ (\_)\ \{n\})$ with W' given by 3.1.3.

2. $L(\sigma)$ is defined so as to have progressively nondecreasing lower bounds for the nodes, as more and more jobs are assigned. For a complete assignment σ, $L(\sigma)$ coincides with the actual weighted sum of completion times for σ.

3. This algorithm is applied to the cluster analysis problem in section 3.3 with appropriate lower bounds. An example illustrating the use of the algorithm is also worked out in the same section.

3.1.3 Identical Machines Problem

The $(n/(m)/F/C_u)$ problem with the machines being identical, i.e. $p_{ij} = p_j$ for all i, is considered in this section. We assume here the jobs are numbered such that $(1,2,\ldots,n)$ is a Mc Naughton's sequence.

Theorem 3.1.2 : Let $B = \{B_i,\ i = 1,2,\ldots,m\}$ be a partition of the set N_q of first q jobs. Let $t_B = \underset{1\le i\le m}{\text{Min}} \sum_{k\in B_i} p_k$. Let $\bar{W}(B)$ be as given by the definition 3.1.2. Then we have

$$\bar{W}(B) \ge \sum_{i=1}^{m} W_i(B_i) + t_B \sum_{j\in \bar{N}_q} u_j + \delta(\bar{N}_q)$$

where $\delta(\bar{N}_q) = \sum_{j\in \bar{N}_q} p_j u_j + \text{Max}[\delta_1(\bar{N}_q), \delta_2(\bar{N}_q)]$

and $\delta_i(\bar{N}_q)$, $i = 1,2$ are given by 3.1.4 and 3.1.5 respectively.

Proof : Consider the $(/\!/\ (m)/F/C_u)$ problem with set of jobs A ($\subseteq N$ when all the jobs are available for processing at time $t \ge 0$. We observe that the weighted sum of completion times for a partition $\{A_1, A_2, \ldots, A_m\}$ of A for this problem is given by,

$$t \sum_{j\in A} u_j + \sum_{i=1}^{m} W_i(A_i)$$

where $W_i(A_i)$ is given by 3.1.1.

We also know $\delta(\bar{N}_q)$ is a lower bound for the weighted sum of completion times corresponding to any partition of jobs in $\bar{N}_q$. Also note that t_B is the earliest instant any job j in $\bar{N}_q$ can be taken up for processing since, the jobs are numbered according to Mc Naughton's sequence. Combining these facts we get the required result.

Remarks

1. Note that $t_B \sum_{j \in \bar{N}_q} u_j$ is nothing but $\sum_{j \in \bar{N}_q} S_j$ in $\upsilon(\bar{N}_q)$ defined earlier. But we could only take $\text{Max}(\upsilon(\bar{N}_q), \delta(\bar{N}_q))$ as a component of the lower bound in the nonidentical machines case.

2. In the nonidentical machines case we cannot assert that t_B is the earliest instant of time at which any job in $\bar{N}_q$ can be taken up for processing. However, we can do so if the jobs can be numbered such that

$$\frac{p_{i1}}{u_1} \leq \frac{p_{i2}}{u_2} \leq \ldots \leq \frac{p_{in}}{u_n} \quad \text{for all } i.$$

Proposition 3.1.1 : If B is an optimal complete partition then $B_i \neq \phi$ for all $1 \leq i \leq m$.

Proof is trivial. Or,

Proof : Suppose B is optimal but B_k is empty. Let B_λ be such that it has at least 2 jobs (there exists such a B_λ as $n > m$). Let $j, s \in B_\lambda$ and $j > s$. Then consider B^* obtained from B as follows

$$B_i^* = \begin{cases} B_i, & \lambda \neq i \neq k \\ \{j\}, & i = k \\ B_\lambda - \{j\}, & i = \lambda \end{cases}$$

Now the weighted completion time of j is reduced by at least $u_j p_s > 0$, and the weighted completion time of no job is increased, a contradiction which proves the result.

This result is not true in the case of nonidentical machines.

Using these results we modify the algorithm given for the nonidentical machines problem. Both the lower bounds and the branching rules are different.

An algorithm for the $(n/(m)/F/C_u)$ problem with identical machines

All partitions are represented by the node of a tree with root at the empty partition denoted by $(\circ)$. Any node in the tree is called a node of order (q, s) if the number of jobs assigned is q and the number of nonempty B_i's is s, in the corresponding partition $B = \{B_i,\ i = 1,2,\ldots,m\}$. We take the s nonempty B_i's to be $B_1, B_2, \ldots, B_s$ without loss of generality.

A node σ of order (q, s), $0 \leq q < n-m+s$, $0 \leq s < m$ is connected by an arc to a node μ of order $(q+1, s+1)$ if the assignment of first q jobs in μ is identical with that of σ and the $q+1^{st}$ job is assigned to $s+1^{st}$ machine. Also a node σ of order (q, s), $0 \leq q < n-m+s$, $0 \leq s \leq m$ is

connected by an arc to each of the s nodes μ of order $(q+1, s)$, where the assignment of the first q jobs in μ is identical with that of σ and the $q+1^{st}$ job is assigned to one of the first s machines. If $q = n-m+s$ and $0 \leq s \leq m$, then a (q, s) order node σ is connected by an arc to a node of order (n, m) which represents a complete assignment, where the first q jobs are assigned as in σ and the remaining $m-s$ jobs are assigned one each to the $(m-s)$ remaining machines.

Let σ denote any node $(i_1,\ldots,i_q)$ or $(\cdot)$ selected for branching. Let σ_k denote the descendant of σ obtained by assigning the $(q+1)^{st}$ job to machine k. Let B and B^k be the partitions corresponding to σ and σ_k respectively.

Let

$$\alpha(\sigma_k) = \sum_{i=1}^{m} W_i(B_i) + u_{q+1}\Big(\sum_{j\in B_k^k} p_j\Big) + t_B\Big(\sum_{j\in \bar{N}_{q+1}} u_j\Big) + \delta(\bar{N}_{q+1}) \quad \ldots 3.1.9$$

Step 0 : Find $\delta(\bar{N}_q)$ for $q = 0,1,2,\ldots,n-1$. Set $L((\cdot)) = \alpha((\cdot)) = \delta(N)$, $\pi = \{(\cdot)\}$ and $Z_o = \infty$ or the weighted sum of completion times of any complete assignment σ_o. Choose node $(\cdot)$ for branching and go to Step 1.

Step 1 : Let the order of the node σ selected for branching be (q, s) and delete this node from π. If $0 \leq q < n-m+s$ and $0 \leq s < m$, use branching rule 1. If $q = n-m+s$ and $s < m$, then use branching rule 2. Otherwise, use branching rule 3.

Branching Rule 1 : Generate the $s+1$ descendants $\sigma_1, \sigma_2, \ldots, \sigma_{s+1}$ of σ. We get s decendants of order $(q+1, s)$ and one of order $(q+1, s+1)$. Go to Step 2.

Branching Rule 2 : Generate the descendant σ_1 of σ by assigning the $q+i^{th}$ job to the $s+i^{th}$ machine, for $1 \leq i \leq m-s$. σ_1 is a node of order (n, m) which is a complete assignment. Go to Step 3 with $L(\sigma_k) = L(\sigma)$.

Branching Rule 3 : Generate the m descendants of σ. Each of them is of order $(q+1, m)$. Go to Step 2.

Step 2 : Calculate $L(\sigma_k)$ by :

$$L(\sigma_k) = \text{Max}[L(\sigma), \alpha(\sigma_k)]$$

and go to Step 3.

The remaining steps (3-8) of the algorithm are same as those in the algorithm given for the nonidentical machines problem in section 3.1.2.

We now illustrate the calculation of the lower bound $L(\sigma)$ *with an example.*

Example 3.1.1 Consider the $(5/(2)/F/C_u)$ problem with the following data taken from Rothkopf (1966). The jobs are numbered according to Mc Naughton's rule.

job	p_j	u_j	p_j/u_j	$p_j\ u_j$
1	1	5	1/5	5
2	2	7	2/7	14
3	3	6	3/6	18
4	2	3	2/3	6
5	4	4	1	16

The following table gives $\delta(\bar{N}_q)$ for $0 \leq q \leq 5$.

q	$\delta(\bar{N}_q)$	$\sum_{j\epsilon\bar{N}_q} u_j$
0	77	25
1	68	20
2	46	13
3	22	7
4	16	4
5	0	0

We give the calculation of $\delta_1(\bar{N}_q)$ and $\delta_2(\bar{N}_q)$ for $q=0$. The other values are similarly calculated. We have $u^*(N) = 3$ and $p^*(N) = 1$.

Arrange p_j's in the nondecreasing order i.e.,

$$1, 2, 2, 3, 4$$

We have $k = [5/2] = 2$ and $r = 1$.

So $\delta_1(N) = 3[2 \times 1 + 1(2+2)] = 18$

Arrange u_j's in nonincreasing order i.e.,

$$7, 6, 5, 4, 3$$

We get

$$\delta_2(N) = 1[2 \times 3 + 1(4+5)] = 15$$

Therefore, $\delta(N) = \sum_{j \in N} u_j p_j + \text{Max}(\delta_1(N), \delta_2(N))$

$$= 59 + \text{Max}(18, 15) = 77.$$

Next we illustrate the calculation of $L(\sigma_k)$ for σ_k with $B_1^k = \{1, 3\}$ and $B_2^k = \{2\}$. We have σ given by $B_1 = \{1\}$ and $B_2 = \{2\}$, also

$$\sum_{i=1}^{2} W_i(B_i) = 19 \quad \text{and} \quad L(\sigma) = 78.$$

Now

$$\alpha(\sigma_k) = 19 + 6(1+3) + 2 \times 7 + 22$$
$$= 79$$

And so $L(\sigma_k) = \text{Max}(78, 79) = 79$.

Thus, after generating 12 nodes, we get the optimal solution as (1,3,4; 2,5) with 85 as weighted sum of completion times.

3.2 Batch Splitting Problem

3.2.0 Introduction :

In this section we consider the $(n/m/R/F_{max})$ problem. All the jobs have to be processed through all the machines once and only once and there are no technological restrictions. A general formulation and a solution method based on Balas' (1969) disjunctive graph approach are suggested. When the jobs are identical, i.e., $p_{ij} = p_i$ for all j, the problem is formulated as a problem of grouping the jobs into m batches and finding an order of processing for each batch so as to minimise the total elapsed time. We then define Latin square arrangement of batches on machines. For any Latin square arrangement we find an optimal partition of the N jobs into m batches. We also prove that by considering only Latin square arrangements we get solutions which are in fact optimal over the set of all

possible schedules, when n is sufficiently large or when n is divisible by m. In addition, we find optimal solutions under the restriction that we use rotation arrangements, when n is not large enough and is not divisible by m.

3.2.1 A general formulation of the $(n/m/R/F_{max})$ problem

There are n jobs to be processed through m machines. Besides making the usual assumptions A.1 through A.7, we assume that

(i) all the n jobs have to be processed through the m machines, once and only once through each machine

(ii) there are no constraints on the order in which a job is to be processed through the machines

and (iii) there are no constraints on the order in which the jobs have to be processed on a machine.

The problem is to find a schedule satisfying the above requirements and minimising the total elapsed time.

Balas (1966, 1969, 1970) gives a disjunctive graph formulation to the job shop sequencing problem $(n/m/J/F_{max})$. The problem given above is same as the $(n/m/J/F_{max})$ problem, excepting that there are no technological ordering restrictions

in the randomly routed shop. This implies that there is freedom of choice in not only the sequence of operations done on a machine but also the sequence of machines through which each job is processed. Balas (1966, 1969) shows that the $(n/m/J/F_{max})$ problem is equivalent to finding a minimaximal path in a disjunctive graph. We give below a corresponding disjunctive graph for the $(n/m/J/F_{max})$ problem.

Definition 3.2.1 :

A disjunctive graph $D = (N,A,B)$ is a (directed) graph (N,A) together with a specified subset B of arcs called disjunctive arcs such that if $(i,j) \in B$ then $(j,i) \in B$.

A path in the disjunctive graph $D = (N,A,B)$ is a path μ in (N,A) such that if $(i,j) \in B$ and $(i,j) \in \mu$ then $(j,i) \notin \mu$.

The notation we use for a disjunctive graph is slightly different from that of Balas.

Consider the disjunctive graph $D = (N, E(\_)B, B)$ defined as follows.

$$N = Z (\_) \{S\} (\_) \{F\}$$

where $Z = \{ \alpha_{ij} : 1 \leq i \leq m, \ 1 \leq j \leq n \}$,

α_{ij} being the operation of processing job j on machine i and S and F are nodes corresponding to two dummy operations 'Start' and 'Finish' as in CPM formulation.

The set of ordinary arcs E and the set of disjunctive arcs B are defined as follows.

$$E = \{(S, \alpha) : \alpha \in Z\} \cup \{(\alpha, F) : \alpha \in Z\}$$

and

$$B = \{(\alpha, \beta) : \alpha, \beta \in Z \text{ and if } \alpha = \alpha_{ij} \text{ and } \beta = \alpha_{k\ell} \text{ then either } i=k \text{ or } j = \ell\}.$$

The disjunctive arc (α, β) expresses the condition that one of the operations α, β must be finished before the other is started.

Now proceeding in the same lines as in Balas (1970) we can show that the $(n/m/R/F_{max})$ problem is equivalent to the problem of finding a minimaximal path in the disjunctive graph $D = (N, E \cup B, B)$. Methods for solving this problem are available [see Balas (1969), Nabashima (1971), Krishnamurthy (1969)].

Next we consider some special cases of the problem.

3.2.2 Identical Jobs and Batch Splitting

Consider the problem stated in section 3.2.1 with the jobs being identical. Let $p_i > 0$ be the processing time for any job on machine i. Call this $(n/m/R/F_{max})$ problem with identical jobs as problem P1. We now consider a related problem.

Problem P2 : Find $(n_1, n_2, \ldots, n_m)$ and m permutations $(\pi_1, \pi_2, \ldots, \pi_m)$ of the numbers $1, 2, \ldots, m$ such that

(i) $n_i \geq 0$, $\sum_{i=1}^{m} n_i = n$, n_i integer

(ii) the k^{th} batch with processing time $n_k p_i$ on the i^{th} machine follows the order of processing through the m machines given by π_i

and (iii) the total elapsed time is minimised.

Problem P2 is problem P1 with further restrictions, because

(i) we require that the n items have to be grouped into m batches only

and (ii) the batches once formed are not to be altered.

We shall show below that these are not really restrictive by exhibiting optimal solutions to P1 which satisfy the restrictions imposed by problem P2.

Consider the following schedule of processing the batches on the machines. Let $\pi_1, \pi_2, \ldots, \pi_m$ be m permutations of $(1,2,\ldots,m)$, and let $L = (\pi_1, \ldots, \pi_m)$ be the $m \times m$ matrix with π_j as the j^{th} column. Each row $(i_1, \ldots, i_m)$ of L gives an assignment of the batches to the machines where the i_k^{th} batch is assigned to machine k. The processing for all assignments specified by a given row of L is taken up at the same time. Also the set of assignments specified by the i^{th} row of L is taken up only after the set of assignments specified by rows $1,2,\ldots,i-1$ are completed.

Definition 3.2.2 :

If L is a latin square, the above schedule of processing is called a Latin square arrangement (LSA).

Let S be the Latin square

$$\begin{pmatrix} 1 & 2 & 3 & \ldots & m \\ m & 1 & 2 & \ldots & m-1 \\ m-1 & m & 1 & \ldots & m-2 \\ \vdots & & & & \vdots \\ 2 & 3 & 4 & \ldots & 1 \end{pmatrix}$$

Then the LSA corresponding to S is called a rotation arrangement.

If we restrict our solutions to Latin square arrangements only, we restrict problem P2 further in that,

(i) we require that the order of processing followed by the batches forms a Latin square

(ii) a row of assignments is taken up for processing only when all the previous rows are completed

and (iii) All assignments specified by a row are taken up at the same time.

Consider now two more problems given below.

Problem P3 : Find a Latin square $L = ((\lambda_{ij}))$ and $(n_1, n_2, \ldots, n_m)$ such that

(i) $\sum_{i=1}^{m} n_i = n, \quad n_i \geq 0$

(ii) $t_i - n_{\lambda_{ij}} p_j \geq 0 \quad$ for $\quad i, j = 1, 2, \ldots, m$

and (iii) $t = \sum_{i=1}^{m} t_i \quad$ is minimised.

Problem P4 is same as Problem P3 with the restriction that n_i's are integers.

Note that problem P4 is same as problem P2 with the restriction that only LSA's are considered.

Lemma 3.2.1 If $[L_1; (n_1, \ldots, n_m)]$ is a solution to problem P3 then $[L_2; (n_1, \ldots, n_m)]$ is also a solution to the problem

when L_2 is obtained from L_1 by a row permutation.

Further if L_2 is obtained from L_1 by a column permutation then L_2 with the corresponding permutation of $(n_1,\ldots,n_m)$ is a solution to the problem.

The proof is trivial and so is omitted.

Theorem 3.2.1 : Any Latin square L and $(n_1,\ldots,n_m)$ where $n_i = n/m$ for all i, solve problem P3 with $t = np'$ where $p' = \underset{j}{\text{Max}}\ p_j$.

Proof : Take,

$$n_i = (n/m) \quad \text{for all } i$$

and let

$$t_i = n/m \times p'.$$

Then the constraints of P3 are satisfied and

$$\sum_{i=1}^{m} t_i = t = np'.$$

Consider any Latin square L. Let $p' = p_h$. Since L is a Latin square, $(\lambda_{1h}, \lambda_{2h},\ldots,\lambda_{mh})$ is a permutation of $(1,\ldots,m)$. Now for any feasible solution to the problem,

$$\sum_i t_i \geq \sum_i n_{\lambda_{ih}} p_h = np'.$$

This completes the proof of the theorem.

Corollary 3.2.1 If $n = mk$ for some integer k, then for any Latin square L, $[L\ ;(k,\ldots,k)]$ solves problems P4 and P2.

when L_2 is obtained from L_1 by a row permutation.

Further if L_2 is obtained from L_1 by a column permutation then L_2 with the corresponding permutation of $(n_1,\ldots,n_m)$ is a solution to the problem.

The proof is trivial and so is omitted.

Theorem 3.2.1 : Any Latin square L and $(n_1,\ldots,n_m)$ where $n_i = n/m$ for all i, solve problem P3 with $t = np'$ where $p' = \underset{j}{\text{Max}}\ p_j$.

Proof : Take,

$$n_i = (n/m) \text{ for all } i$$

and let

$$t_i = n/m \times p'.$$

Then the constraints of P3 are satisfied and

$$\sum_{i=1}^{m} t_i = t = np'.$$

Consider any Latin square L. Let $p' = p_h$. Since L is a Latin square, $(\lambda_{1h}, \lambda_{2h},\ldots,\lambda_{mh})$ is a permutation of $(1,\ldots,m)$. Now for any feasible solution to the problem,

$$\sum_i t_i \geq \sum_i n_{\lambda_{ih}} p_h = np'.$$

This completes the proof of the theorem.

Corollary 3.2.1 If $n = mk$ for some integer k, then for any Latin square L, $[L\ ;(k,\ldots,k)]$ solves problems P4 and P2.

This corollary follows from theorem 3.2.1 and the fact that the total elapsed time for any schedule for problem P1 is at least np' since each job has to go through the machine for which the processing time is p'.

Theorem 3.2.2 : Suppose $n = mk+r$, $0 < r \leq m-1$ and $p_1 \leq p_2 \leq \cdots \leq p_{m-1} < p_m$. Let Q be the smallest integer such that

$$\frac{p_{m-1}}{p_m} \leq \frac{Q-1}{Q} .$$

If $k \geq Q-1$ then $[L; (n_1, \ldots, n_m)]$ is a solution to P4 and P2 for any Latin square L with

$$n_i = k+1 \quad \text{if} \quad i \leq r$$

$$n_i = k \quad \text{if} \quad i \geq r+1.$$

Also the corresponding total elapsed time

$$t = np_m.$$

Proof : Consider the solution

$$n_i = k+1 \quad \text{if} \quad i \leq r$$

$$n_i = k \quad \text{if} \quad i \geq r+1.$$

Note that

$$(k+1)\, p_m \geq (k+1)\, p_i \geq k\, p_i ,$$

and since $k \geq Q - 1$, we have,

$$k\, p_m \geq (k+1) p_i \quad \text{for} \quad i \neq m$$

Thus t_i equals $(k+1)\, p_m$ if m occurs in the first r places of the i^{th} row of L and t_i equals $k\, p_m$ otherwise.

Therefore, t_i equals $(k+1)\, p_m$ exactly r times and t_i equals $k\, p_m$ exactly $(m - r)$ times.

Thus

$$t = \sum_{i=1}^{m} t_i = (mk + r)\, p_m = np_m.$$

Hence the theorem is proved.

The above results show that the additional restrictions imposed on problem P1 to get problems P2, P3 or P4 are not really restrictive when n is divisible by m or n is sufficiently large.

We now solve problem P4 fixing $L = S$, when the hypothesis of theorem 3.2.2 does not hold. Let P4 with L fixed as S be designated as problem P5. Let

(i) $n = mk+r, \quad 0 < r \leq m-1$

(ii) $p_1 \leq p_2 \leq \ldots \leq p_m$

and (iii) h be the maximum integer, $1 \leq h \leq m-1$ for which

$$\frac{k}{k+1} \leq \frac{p_{m-h}}{p_m}.$$

Then we have

Lemma 3.2.2 : Suppose $r \leq m-h$. Then for any solution $(n_1,\ldots,n_m)$ to problem P5,

$$t \geq (k+1) \sum_{j=1}^{h} p_{m-j} + (n - hk)\, p_m .$$

Proof : Let $n_o = \text{Max}(n_1,\ldots,n_m)$, then $n_o \geq k+1$. Let s be the number of n_i's greater than k. Evidently $s \geq 1$.

Case 1 $m - s \geq h$.

Let $P = \{i \mid n_{\lambda_{im}} \leq k\}$.

Define $q = \underset{i \in P}{\text{Min}} \{j \mid n_{\lambda_{ij}} \geq k+1,\ n_{\lambda_{iw}} \leq k \text{ for all } w > j\}$,

and let the above minimum be attained for $v \in P$. If $q < m-h$, then since $\lambda_{vq} = \lambda_{m-h-q+v+i,m-h+i}$, it follows that for $i = 0,1,\ldots,(h-1)$, in row $m-h-q+v+i$ a batch of size $\geq k+1$ is assigned to machine $m-h+i$. If a row number exceeds m, its residue modulo m is considered.

Therefore,

$$t_{m-h-q+v+i} \geq (k+1)\, p_{m-h+i}, \quad i = 0,1,\ldots,h-1.$$

Also in these rows machine m is assigned to a batch of size $\leq k$. Therefore there are at least $(n-hk)$ jobs to be processed through machine m in the other row assignments.

Thus $t \geq (k+1) \sum_{i=o}^{h-1} p_{m-h+i} + (n-hk)\, p_m$.

If $q > m-h$, since $\lambda_{vq} = \lambda_{v+i,\,q+i}$ it follows that for $i = 0,1,\ldots,m-1-q$, in row $v+i$, a batch of size $\geq k+1$ is assigned to machine $q+i$.

Therefore,

$$t_{v+i} \geq (k+1)\, p_{q+i}, \quad i = 0,1,\ldots,m-1-q.$$

Consider now the other $m-s-(m-q) = q-s > q-m+h$ rows in P. In each of them, a batch of size $\geq k+1$ is assigned to a machine with index at least q. Thus the contribution to t from any $q-m+h$ of these rows $\geq (k+1)\,(p_{m-h} + \ldots + p_{q-1})$.

We thus obtain

$$t \geq (k+1) \sum_{i=o}^{h-1} p_{m-h+i} + (n-hk)p_m .$$

Case 2 $m-s < h$.

Let q, v, be as defined before. Note that q cannot be less than $m-h$, because $m-s < h$ and that $q \geq s$. So by the definition of q, in each of the rows of P, a batch of size $\geq (k+1)$ is assigned to a machine with index at least q. Also in rows $v+i$, for $i = 0,1,\ldots,m-1-q$, a batch of size $\geq k+1$ is assigned to the machine $q+i$. Observe that there are s rows in which a batch of size $\geq k+1$ is assigned to machine m.

Thus at least $s(k+1)$ jobs are to be processed on the machine m in the rows not in P. So

$$t \geq (k+1) \sum_{i=1}^{m-s} p_{m-i} + s(k+1)p_m$$

$$= (k+1) \sum_{i=1}^{m-s} p_{m-i} + (n-hk)p_m - (n-hk)p_m + s(k+1)p_m.$$

Now using $n = mk+r$, we get

$$t \geq (k+1) \sum_{i=1}^{m-s} p_{m-i} + (n-hk)p_m + [(h+s-m)k + (s-r)]p_m$$

$$\geq (k+1) \sum_{i=1}^{m-s} p_{m-i} + (n-hk)p_m + [h-(m-s)]\ (k+1)p_m$$

since $s-r \geq h+s-m$ by hypothesis

$$\geq (k+1) \sum_{i=1}^{h} p_{m-i} + (n-hk)p_m.$$

This completes the proof of lemma 3.2.2.

Lemma 3.2.3 : Let $r > m-h$ where r,m,h are as defined in (i), (ii) and (iii) preceding lemma 3.2.2. Then for any solution $(n_1,\ldots,n_m)$ to problem P5,

$$t \geq (k+1) \sum_{i=1}^{m-r} p_{m-i} + r(k+1)p_m.$$

Proof : Let P,s,q,v be as defined in the proof of lemma 3.2.2.

Case (i) $m-s \geq h$.

In this case we have as in Case 1 of the proof of lemma 3.2.2,

$$t \geq (k+1) \sum_{i=o}^{h} p_{m-i} + (n-kh)p_m$$

$$\geq (k+1) \sum_{i=1}^{m-r} p_{m-i} + (h-m+r)\, kp_m + (n-kh)p_m$$

since $(k+1)p_{m-i} \geq kp_m$ when $i \leq h$

$$= (k+1) \sum_{i=1}^{m-r} p_{m-i} + r(k+1)p_m$$

as required.

Case (ii) $m-s < h$ and $s \leq r$.

We get as in Case 2 of the proof of lemma 3.2.2, in each of the rows of P a batch of size $\geq k+1$ assigned to a machine λ with $\lambda \geq q$. Also in rows $v+i$, for $i = 0,1,\ldots,m-1-q$, a batch of size $\geq k+1$ is assigned to the machine $q+i$.

Now at most $(m-s)k$ items are processed on machine m in all the rows in P. Therefore there are at least $(n-(m-s)k)$ jobs to be processed through machine m in the other rows of assignments.

Thus

$$t \geq (k+1) \sum_{i=1}^{m-s} p_{m-i} + (n-(m-s)k)p_m$$

$$\geq (k+1) \sum_{i=1}^{m-r} p_{m-i} + (r-s)kp_m + (r+sk)p_m$$

since $(k+1)\, p_{m-i} \geq kp_m$ when $i \leq h$

$$= (k+1) \sum_{i=1}^{m-r} p_{m-i} + r(k+1)p_m$$

as required.

Case (iii) $m - s < h$ and $s > r$

As in Case 2 of the proof of lemma 3.2.2, we get

$$t \geq (k+1) \sum_{i=1}^{m-s} p_{m-i} + s(k+1)p_m$$

$$= (k+1) \sum_{i=1}^{m-r} p_{m-i} + r(k+1)p_m - (k+1) \sum_{i=m-s+1}^{m-r} p_{m-i} + (s-r)(k+1)p_m$$

Now the required inequality follows as $p_{m-i} \leq p_m$.

This completes the proof of the lemma.

Theorem 3.2.3 : If $n = mk+r$, $1 \leq r \leq m-1$ and $L = S$, the rotation arrangement, then L and $(n_1,\ldots,n_m)$ such that

$$n_i = k+1 \quad \text{if} \quad i \leq r$$

$$= k \quad \text{if} \quad i \geq r+1$$

Solve the problem P5.

Proof: By theorem 3.2.2 and the lemmas 3.2.2 and 3.2.3 and noting the fact that the lower bounds given by the lemmas are actually attained for this solution the result is immediate.

Next, we show that rotation arrangements are not in general optimal solutions even for problem P4, by the $(10/4/R/F_{max})$ problem with the data given below.

Example 3.2.1 : Consider the $(10/4/R/F_{max})$ problem with $p_1 = 1$, $p_2 = 2$, $p_3 = 14$, $p_4 = 15$ as processing times for any job on machines 1,2,3 and 4 respectively. Note that $k = 2$, $\frac{k}{k+1} = \frac{2}{3} < \frac{p_3}{p_4} = \frac{14}{15}$ and $\frac{2}{3} > \frac{p_2}{p_4} = \frac{2}{15}$, so $h = 1$. Hence theorem 3.2.3 is applicable and $L = S$ and $n_1 = 3$, $n_2 = 3$, $n_3 = 2$ and $n_4 = 2$ give an optimal solution under the restriction $L = S$.

Now $t_1 = 30$, $t_2 = 42$, $t_3 = t_4 = 45$,

so $t = 162$.

Now consider the Latin square arrangement given by

$$L = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \\ 3 & 4 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{pmatrix}$$

and n_1, n_2, n_3 and n_4 as given above. We then have $t_1 = 30 = t_2$ and $t_3 = t_4 = 45$. Hence $t = 150 < 162$. So $L = S$ is not optimal for problem P4.

3.3 Cluster Analysis

3.3.0 Introduction :

In this section we deal with the cluster analysis problem which may be formulated as follows.

Let $N = \{1, 2, \ldots, n\}$ as usual. Suppose we are given for each $i \in N$, a vector $X_i = (X_{1i}, X_{2i}, \ldots, X_{zi}) \in R^z$.

Let $d_{ij} = |X_i - X_j| = [\sum_{k=1}^{z} (X_{ki} - X_{kj})^2]^{\frac{1}{2}}$ and for any $A \subseteq N$, define

$$\tau(A) = \begin{cases} \frac{1}{\lambda} \sum_{\substack{i<j \\ i,j \in A}} d_{ij}^2 & \text{for } \lambda \geq 2 \\ 0 & \text{otherwise} \end{cases}$$

where λ is the number of elements in A.

The cluster analysis problem is to find a partition $J = (J_1, \ldots, J_m)$ of N such that

$$C'(J) = \sum_{i=1}^{m} \tau(J_i) \text{ is minimised.}$$

An extensive survey of work done on this problem up to 1969 is available in Jenson (1969).

In section 3.3.1 we derive the main results and in section 3.3.2 we give an example to illustrate how the algorithm developed in section 3.1.3 can be used for this problem.

3.3.1 Main Results

First we state the following well known lemma.

Lemma 3.3.1 : Let $X \in R^z$, then

$$\sum_{i=1}^{n} |X - X_i|^2 \geq \sum_{i=1}^{n} |\bar{X} - X_i|^2 = \frac{1}{n} \sum_{1 \leq i < j \leq n} |X_i - X_j|^2$$

where $\bar{X} = \frac{1}{n}(X_1 + X_2 + \ldots + X_n)$.

Lemma 3.3.2 : For any two disjoint subsets A_1 and A_2 of N we have,

$$\tau(A_1 \cup A_2) \geq \tau(A_1) + \tau(A_2).$$

This lemma is easy to prove using lemma 3.3.1.

Before proceeding further with the main problem, we consider the following problem. For convenience we denote $\binom{n}{2}$ by $\overset{\Lambda}{n}$.

Let $I = \{1, 2, \ldots, \overset{\Lambda}{n}\}$. In what follows, G denotes $(G_1, \ldots, G_n)$ where $G_i \subseteq I$ for all i and $G_i \cap G_j = \phi$ if $i \neq j$.

Problem 2. Given n, m and nonnegative real numbers

$$a_1 \leq a_2 \leq \ldots \leq a_{\overset{\Lambda}{n}},$$

find $n_1, n_2, \ldots, n_m$ and G such that

(i) $\sum_{i=1}^{m} n_i = n$

(ii) n_i are positive integers

(iii) G_i has $\hat{n}_i$ elements for $1 \leq i \leq m$

and $C(G) = \sum_{i=1}^{m} \frac{1}{n_i} \sum_{j \in G_i} a_j$ is minimized.

Definition 3.3.1

An alternative denoted by $\mathbf{P}$ is a vector $(n_1, n_2, \ldots, n_m)$ of positive integers such that

$$\sum_{i=1}^{m} n_i = n \quad \text{and} \quad n_1 \leq n_2 \leq \ldots \leq n_m.$$

Let $\underline{P}_{n,m} = \{ P : P$ is an alternative corresponding to n and $m \}$.

Definition 3.3.2

G is called a Grouping induced by $P \in \underline{P}_{n,m}$ if the number of elements in G_i is $\hat{n}_i$ for all i, where $P = (n_1, n_2, \ldots, n_m)$.

Let $\underline{G}(P) = \{G : G$ is induced by $P\}$

for a given $P \in \underline{P}_{n,m}$.

Clearly Problem 2 is equivalent to

$$\min_{P \in \underline{P}_{n,m}} \left[\min_{G \in \underline{G}(P)} C(G) \right]$$

First we consider $\min_{G \in \underline{G}(P)} C(G)$ and give an optimal solution for a given P and then consider Problem 2.

In the following we write $\underline{P}$ for $\underline{P}_{n,m}$.

Lemma 3.3.3 : Given $P \in \underline{P}$, $G^*(P)$ defined by

$$G_i^*(P) = \left\{ j : \sum_{k=1}^{i-1} \hat{n} + 1 \leq j \leq \sum_{k=1}^{i} \hat{n}_k \right\} \quad \text{for} \quad 1 \leq i \leq m$$

minimizes $C(G)$ over $\underline{G}(P)$.

The lemma follows trivially from the fact : Given $a_1, a_2, \ldots, a_k$ and $b_1, b_2, \ldots, b_k$ nonnegative, $\sum_{i=1}^{k} a_{j_i} b_{j_i}$ is minimised when $j_1, \ldots, j_k$ is such that $a_{j_1} \leq \ldots \leq a_{j_k}$ and $b_{j_1} \geq \ldots \geq b_{j_k}$.

Lemma 3.3.4 : Let $P = (n_1, n_2, \ldots, n_m)$ and $P' = (n_1', n_2', \ldots, n_m')$ be alternatives such that

(i) $1 = n_k \leq n_k' \leq n_{k+1}$ and

$n_s' \geq n_{s-1}$ for some k and s with $1 \leq k < s \leq m$

and (ii) $n_j' = n_j$ for $s \neq j \neq k$.

Then we have

$$C(G^*(P)) \geq C(G^*(P'))].$$

Proof : Let $\lambda = n_k' - n_k = n_s - n_s'$.

Consider $G \in \underline{G}(P')$, defined as follows

$G_i = G_i^*(P)$ if $s \neq i \neq k$,

$G_k = G_k^*(P) \cup H$

where H = the set of first $\overset{\wedge}{\lambda} + n_k \lambda$ terms from $G_s^*(P)$

and G_s = the set of first $\overset{\wedge}{n_s'}$ terms from $G_s^*(P) - H$.

Note that this is possible by the properties of n_k, n_k', n_s and n_s' given in the hypothesis.

Let $F = G_s^*(P) - (G_s \cup H)$.

Now $C(G^*(P)) - C(G) = \frac{1}{n_k} \sum_{j \in G_k^*(P)} a_j - \frac{1}{n_k'} \sum_{j \in G_k} a_j + \frac{1}{n_s} \sum_{j \in G_s^*(P)} a_j$

$$- \frac{1}{n_s'} \sum_{j \in G_s} a_j$$

$$= \left(\frac{1}{n_k} - \frac{1}{n_k'}\right) \sum_{j \in G_k^*(P)} a_j + \left(\frac{1}{n_s} - \frac{1}{n_k'}\right) \sum_{j \in H} a_j$$

$$+ \left(\frac{1}{n_s} - \frac{1}{n_s'}\right) \sum_{j \in G_s} a_j + \frac{1}{n_s} \sum_{j \in F} a_j$$

Let $\underset{j \in G_s}{\text{Max}}\ a_j = a'$. Then we have $\underset{j \in F}{\text{Min}}\ a_j \geq a' \geq \underset{j \in H}{\text{Max}}\ a_j$.

Thus as the first term is zero in the expression for $C(G^*(P)) - C(G)$, we get

$$C(G^*(P)) - C(G) \geq \left(\frac{1}{n_s} - \frac{1}{n_{k+\lambda}}\right) \left[\overset{\Lambda}{\lambda} + n_k \lambda \right] a'$$

$$+ \left(\frac{1}{n_s} - \frac{1}{n_s'}\right) \overset{\Lambda}{n_s'} \; a'$$

$$+ \frac{1}{n_s} \left[\overset{\Lambda}{n_s} - \overset{\Lambda}{n_s'} - \left(\overset{\Lambda}{\lambda} + n_k \lambda \right) \right] a'$$

$$= \frac{a'\lambda}{2} \left[1 - \frac{(\lambda - 1 + 2n_k')}{n_k + \lambda}\right]$$

$$= 0 \qquad \text{after simplification.}$$

Thus $C(G^*(P)) \geq C(G)$.

By definition $C(G^*(P')) \leq C(G)$ for any $G \in \underline{G}(P')$.

Hence the result follows.

Remark : This lemma shows that when $n > 2m$, if any $P \in \underline{P}$ has some n_i equal to one then there exists a $P' \in \underline{P}$ such that $C(G^*(P')) \leq C(G^*(P))$. Further when $n \leq 2m$, P^* is given by

$$n_i = \left[\frac{n}{m}\right] \quad \text{for} \quad 1 \leq i \leq m-s$$

$$= \left[\frac{n}{m}\right] + 1 \quad \text{for} \quad m-s+1 \leq i \leq m$$

where $s = n - [\frac{n}{m}]m$ is such that $G^*(P^*)$ is optimal for problem 2.

Going back to the main problem, consider any subset $A \subseteq N$ having at least m elements. Consider d^2_{ij} for $i, j \in A$, $i < j$. Arrange them in nondecreasing order and rename them as

$$a_1, a_2, \ldots \quad \text{i.e.,} \quad a_k = d^2_{i_k j_k}$$

for some i_k and $j_k \in A$ for $k = 1, 2, \ldots, \overset{\wedge}{\lambda}$ where λ is the number of elements in A ($\lambda \geq m$).

Let P^* be such that

$$C(G^*(P^*)) \leq C(G^*(P)) \quad \text{for all} \quad P \in \underline{P}_{\lambda, m}$$

Then we have the following two lemmas.

Lemma 3.3.5 : $C'(J^*) \geq C(G^*(P^*))$ for any optimal partition J^* of A.

Proof : We have

$$C'(J^*) = \sum_{k \to n_k \geq 2} \frac{1}{n_k} \left(\sum_{\substack{i<j \\ i,j \in J^*_k}} d^2_{ij} \right)$$

where n_k is the number of elements in J^*_k.

Also without loss of generality we assume

$$n_1 \leq n_2 \leq \ldots \leq n_m$$

Note that we can choose J^* such that no $n_i = 0$ (by lemma 3.3.2).

Let $P = (n_1, n_2, \ldots, n_m) \in \underline{P}_{\lambda,m}$. Let $G \in \underline{G}(P)$ be defined by putting r in G_k iff $i,j \in J_k^*$ where $a_r = d_{ij}^2$.

Then, $C'(J^*) = C(G)$.

But $C(G^*(P^*)) \leq C(G^*(P)) \leq C(G)$ and the lemma follows immediately.

Lemma 3.3.6 : $C(G^*(P^*)) \geq (\frac{b-1}{2})(\sum_{i=1}^{m-r} a_i) + (\frac{b}{2}) \sum_{i=m-r+1}^{m} a_i$

where $b = [\frac{\lambda}{m}]$, $r = \lambda - [\frac{\lambda}{m}]m$.

Proof: First let $\lambda > 2m$ we may take that no n_i is equal to 1 in P^* since otherwise we can get an optimal P' satisfying this condition. Therefore, $G_i^*(P^*)$ will have at least one element, for all i.

Now $C(G^*(P^*)) \geq \sum_{i=1}^{m} \frac{n_i(n_i - 1)}{2n_i} a_i$ ... (**)

(since all elements in $G_i^*(P^*) \geq i$)

$$\text{R.H.S. of (**)} = \frac{1}{2} \sum_{i=1}^{m} (n_i - 1)\, a_i .$$

Since $\sum_{i=1}^{m} n_i a_i$ is minimised subject to $\sum_{i=1}^{m} n_i = \lambda$ and $n_1 \leq n_2 \leq \ldots \leq n_m$ when n_i's differ by at most 1 (note that $a_1 \leq a_2 \leq \ldots \leq a_m$), we have the required result.

Now let $\lambda \leq 2m$. From the remark following lemma 3.3.4 we have the required result and in fact equality holds..

Let $\delta(A)$ be the lower bound for $C(G^*(P^*))$ given by the lemma 3.3.6.

Call any partition J of the set $N_q = \{1, 2, \ldots, q\} \subseteq N$ a partial partition when $q < n$.

Let a completion of a partial partition be as given by definition 3.1.1.

Lemma 3.3.7 : Let J be a partition of the set N_q of the first q elements of N. Let $\bar{C}(J)$ be the minimum over all completions of J. Then

$$\bar{C}(J) \geq C'(J) + \delta(\bar{N}_q)$$

where $\delta(\bar{N}_q)$ is as given by lemma 3.3.6.

This result follows easily from lemmas 3.3.2, 3.3.5 and 3.3.6.

Lemma 3.3.8 : Let $J = (J_1, \ldots, J_m)$ be a partial partition of N_q with the number of elements in $J_k = q_k$. Let $S(J_k) = q_k \ \tau(J_k)$ and $S(J_k ; j) = \sum_{i \in J_k} d_{ij}^2$.

Let $\mu_j = \text{Max}\{0, \ \underset{1 \le k \le n}{\text{Min}} \ [\frac{S(J_k; j) + S(J_k)}{j - q + q_k} - \frac{S(J_k)}{q_k}]\}$

and $\upsilon(\bar{N}_q) = \sum_{j \in \bar{N}_q} \mu_j$

Then $\bar{C}(J) \ge C'(J) + \upsilon(\bar{N}_q)$.

Proof: Observe that μ_j is the minimum contribution from j to C' of any completion of J (when j is grouped with any of the J_k).

So, $\upsilon(\bar{N}_q) = \sum_{j \in \bar{N}_q} \mu_j$ gives a lower bound on the contribution to C' from the elements in $\bar{N}_q$ in any completion of J. Therefore, using lemma 3.3.2 we get,

$$\bar{C}(J) \ge C'(J) + \upsilon(\bar{N}_q)$$

as required.

Combining lemmas 3.3.7 and 3.3.8, we have the following theorem.

Theorem 3.3.1: $\bar{C}(J) \ge C'(J) + \text{Max}[\delta(\bar{N}_q), \ \upsilon(\bar{N}_q)]$.

We get an algorithm for solving the cluster analysis problem by using the algorithm given for the $(n/(m)/F/C_u)$ problem with identical machines in section 3.1 with the lower bound given by the above theorem.

We now work out an example to illustrate the use of this algorithm.

Example 3.3.1

The following table gives the d_{ij}^2 for $1 \leq i < j \leq n$

i \ j	1	2	3	4	5	6
1	-	8	9	4	7	7
2		-	3	7	4	5
3			-	8	3	5
4				-	7	6
5					-	6
6						-

We are interested in grouping these 6 items into 2 groups so as to minimize the combined within groups sums of squares. The tree generated is given in Fig. 3.3.1. Optimal clustering is given by (1, 4) and (2,3,5 and 6) with combined within groups sums of squares, 8.5.

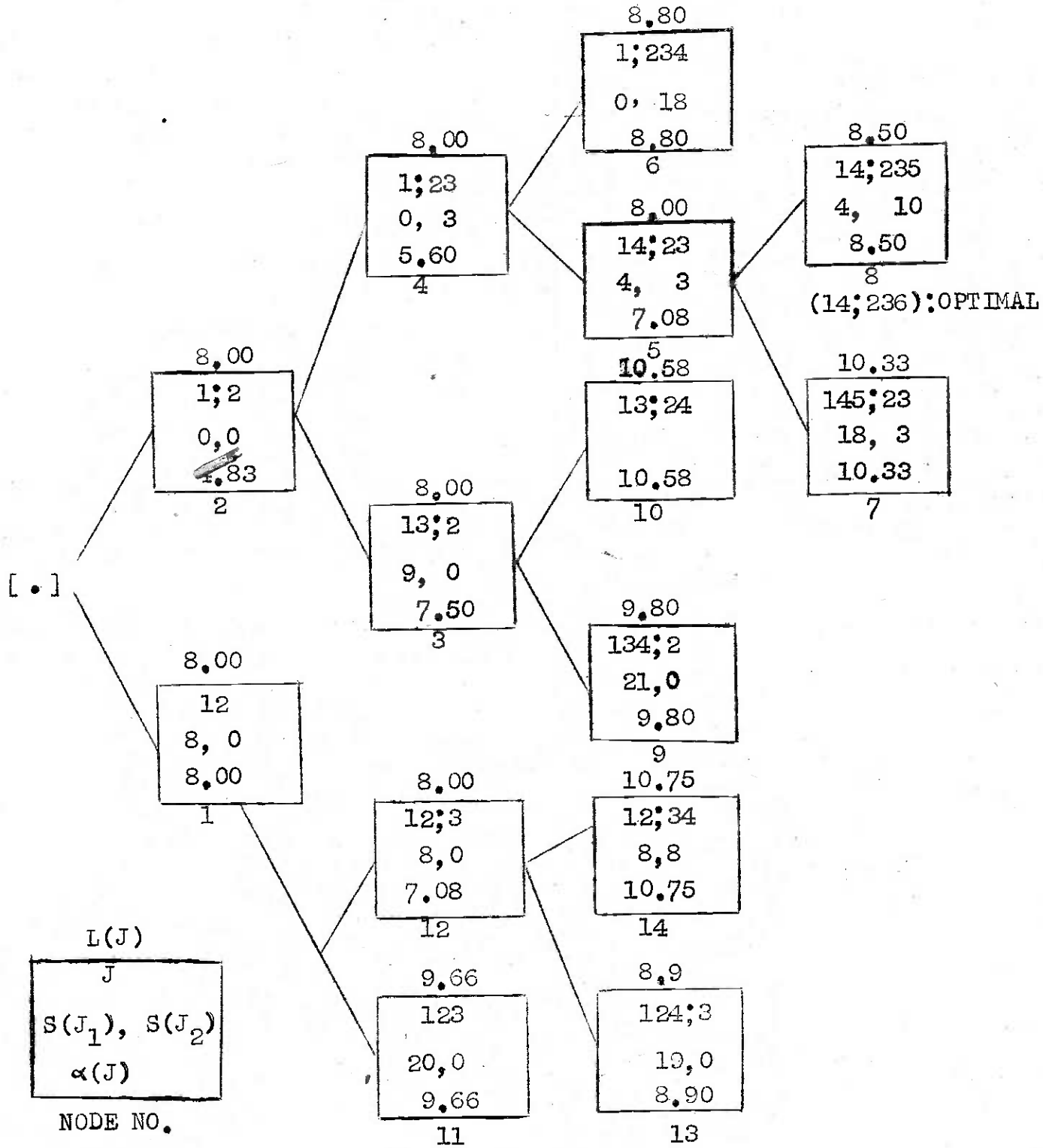


Figure 3.3.1

The calculation of $\beta(\bar{N}_q)$ is shown for $q = 2$. Now, $\bar{N}_q = \{3,4,5,6\}$. Arranging the corresponding d_{ij}^2 in non-decreasing order we get a_j's as :

$$3,5,6,6,7,8.$$

As $b = [\frac{n-q}{m}] = [\frac{4}{2}] = 2$ and $r = 0$,

$$\delta(\bar{N}_q) = \frac{1}{2}(3 + 5) = 4.0 .$$

Similarly $\delta(\bar{N}_q)$ is calculated for other values of q and they are given below :

q	$\delta(\bar{N}_q)$
0	8.00
1	5.55
2	4.00
3	3.00
4,5,6	0.00

Next we show how $\upsilon(\bar{N}_q)$ is calculated for the partial partition $J = (\{1,3\}, \{2,4\})$.

We find, $S(J_1) = 9$ and $S(J_2) = 7$

$$C'(J) = 4.5 + 3.5 = 8.0$$

Now

$\upsilon(\bar{N}_q) = \mu_5 + \mu_6$ is obtained as follows :

$$\mu_5 = \text{Max}[0 \; ; \; \text{Min}(\frac{10 + 9}{3} - 4.5 \; ; \; \frac{11 + 7}{3} - 3.5)]$$

$$= 1.83$$

and $$\mu_6 = \text{Max}[0 \; ; \; \text{Min}(\frac{12 + 9}{4} - 4.5 \; ; \; \frac{11 + 7}{4} - 3.5)]$$

$$= 0.75$$

Thus, $\upsilon (\overline{N}_q) = 2.58.$

So

$$\overline{C}(J) \geq 8.0 + \text{Max}[0, 2.58]$$

$$= 8.0 + 2.58 = 10.58.$$

Using the algorithm, 14 nodes were generated to obtain an optimal complete partition.

LIST OF SYMBOLS

Symbol	Stands for
N	the set of jobs (finite)
n	number of elements in N
N_q	the set of first q jobs of N
$\bar{N}_q$	$N - N_q$
m	number of stages (with one machine in each stage) (m finite)
(m)	number of machines in one stage
F	Flow Shop
R	Randomly routed shop
J	Job Shop
F_{max}	total elapsed time
$\bar{F}$	mean flow time
$\bar{C}$	average completion time
T_u	weighted tardiness
$u(T)$	a real valued function of job tardiness
C_u	weighted sum of completion times
$\bar{T}$	average completion time
f	general objective function
$\bar{p}, \sigma, \mu, \lambda$	permutations of $(1,2,\ldots,n)$ or permutations of any subset of N with distinct elements

Symbol	Stands for
CBS-rule	contiguous binary switching rule
BS-rule	binary switching rule
C_j	completion time of job j
p_{ij}	processing time for job j on machine i (≥ 0)
u_j	penalty rate for job j (≥ 0)
d_j	due date for job j (≥ 0)
T_j	tardiness for job j = $\text{Max}(0, C_j - d_j)$
$\cup$	set operation, union
$\cap$	set inclusion
$\subseteq$	set inclusion
$i \leftarrow j$	job i 'precedes' job j
$\binom{n}{2}$	n_{C_2}
$\forall$	for every
iff	if and only if
$\ni$	such that

REFERENCES

ARTHANARI, T.S. and RAMAMURTHY, K.G. (1970), "A Branch and Bound Algorithm for Sequencing n Jobs on m Parallel Processors", Opsearch, vol.7, no.3.

ARTHANARI, T.S. and RAMAMURTHY, K.G. (1971), "An Extension of Two Machines Sequencing Problem", Opsearch, vol. 8, no.1.

ARTHANARI, T.S. and MUKHOPADHYAY, A.C. (1971), "A Note on a Paper by W. Szwarc", Nav. Res. Log. Quar., vol.18, no.1.

ASHOUR, S., (1970), "A Statistical Analysis of Production Scheduling Systems", J. Jap. Op. Res. Soc., vol. 12, no.2.

BAGGA, P.L. and CHAKRAVARTHI, N.K. (1968), "Optimal m Stage Production Schedule", J. Can. Op. Res. Soc., vol.6.

BAGGA, P.L. and MITTAL (1973), "An Efficient Algorithm for Two Stage Sequencing Problem with Two Parallel Machines at the First Stage", Opsearch, vol. 10.

BAKSHI, M.S. and ARORA, S.R. (1969), "The Sequencing Problem", Man. Sci., vol.16, no.4.

BALAS, E. (1969), "Machine Sequencing via Disjunctive Graphs : An Implicit Enumeration Algorithm", Opn. Res., vol.17, no.6.

BALAS, E. (1966), "Finding a Minimaximal Path in a Disjunctive PERT Network", Theory of Graphs, International Symposium, Rome. Dunod, Paris; Gordon and Breach, New York.

BALAS, E. (1970), "Project Scheduling with Resource Constraints", Applications of Mathematical Programming Techniques, edited by E.M.L. Beale, Reprint no. 485, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

BELLMAN, R. (1956),"Some Mathematical Aspects of Scheduling Theory", J. Soc. Ind. and Appl. Math., vol.4, no.3.

BOWMAN, E.H. (1959), "The Schedule-Sequencing Problem", Opn. Res., vol. 7, no.5.

BROWN, A.P.G. and LOMNICKI, Z.A. (1966), "Some Applications of the Branch and Bound Algorithm to the Machine Scheduling Problem", Opn. Res. Quarterly, vol. 17, no.2.

CHARLTON, J.M. and DEATH, C.C. (1970), "A Generalized Machine-Scheduling Algorithm", Opn. Res. Quarterly, vol. 21, no.1.

DANTZIG, G.B. (1960), "A Machine-Job Scheduling Model", Man. Sci., vol. 6, no.2.

DAY, J.E. and HOTTENSTEIN, M.I. (1970), "Review of Sequencing Research", Nav. Res. Log. Quart., vol. 17, no.1.

DEATH, C.C. and CHARLTON, J.M. (1970) - see CHARLTON, J.M. and DEATH, C.C. (1970).

DUDEK, R.A. and TEUTON, O.F., Jr. (1964), "Development of M-stage Decision Rule for Scheduling n Jobs Through m machines", Opn., Res., vol. 12, no.3.

EASTMAN, W.L., EVEN, S. and ISAACS, I.M. (1964), "Bounds for the Optimal Scheduling of n jobs on m Processors", Man. Sci., vol.11, no.2.

EDWARD, A.W. and CAVALLI-SFORZA.(1965), "A Method for Cluster Analysis", Biometrica, vol. 21.

ELMAGHRABY, S.E. (1968), "The Machine Sequencing Problem - Review and Extensions", Nav. Res. Log. Quart., vol. 15, no.2.

ELMAGHRABY, S.E. (1968'), "The One Machine Sequencing Problem with Delay Costs", J. Ind. Eng., vol. XIX, no.2.

ELMAGHRABY, S.E. (1968a), "The Sequencing of Related Jobs", Nav. Res. Log. Quart., vol.15, no.1.

ELMAGHRABY, S.E.(1971), "A Graph Theoretic Interpretation of the Sufficiency Conditions for the Contiguous-Binary Switching (CBS) Rule", Nav. Res.Log. Quart., vol. 18, no.3.

FISHER, M.L. (1973), "Optimal Solution of Scheduling Problem Using Lagrange Multipliers : Part I", Opn. Res., vol.21, no.5.

FLORIAN, M., TREPANT, P. and McMAHON, G. (1971), "An Implicit Enumeration Algorithm for the Machine Sequencing Problem", Man. Sci., vol.17, no. 12.

GUPTA, J.N.D. and WALVEKAR, A. (1969), "Sequencing n Jobs on M Parallel Processors", Opsearch, vol. 6.

GUPTA, J.N.D. (1971), "Optimal Flowshop Scheduling with Due Dates and Penalty Costs", J. Jap. Opn. Res. Soc., vol. 14.

GUPTA, J.N.D. (1971a), "Economic Aspects of Production Scheduling Systems", J. Jap. Opn. Res. Soc., vol. 13.

GUPTA, J.N.D. (1971'), "An Improved Combinatorial Algorithm for the Flow Shop Scheduling Problem", Opn. Res., vol. 19, no.7.

HAMILTON EMMONS (1969), "One-Machine Sequencing to Minimise Certain Functions of Job Tardiness", Opn. Res., vol. 17, no.4.

HELD, M. and KARP, R.M. (1962), "A Dynamic Programming Approach to Sequencing Problems", J. Soc. Ind. and Appl. Math., vol.10, no.2.

HELLER, J. (1960), "Some Numerical Experiments for an M x J Flow Shop and Its Decision Theoretical Aspects", Opn. Res., vol.8, no.2.

IGNALL, E. and SCHRAGE, L. (1965), "Application of the Branch and Bound Technique to Some Flow Shop Scheduling Problems", Opn. Res., vol. 13, no.3.

JACK EDMONDS (1965), "Paths, Trees and Flowers", Can. Jour. of Maths., vol.17, no.3.

JACKSON, J.R. (1956), "An Extension of Johnson's Results on Job-Lot Scheduling", Nav. Res. Log. Quart., vol.3, no.3.

JENSON, R.E. (1969), "A Dynamic Programming Algorithm for Cluster Analysis", Opn. Res., vol. 17, no.6.

JOHN, G. RAU (1971), "Minimising a Function of Permutations of n Integers", Opn. Res., vol. 19, no.1.

JOHNSON, S.M. (1954), "Optimal Two-and Three-stage Production Schedules with Set-up Times Included", Nav. Res. Log. Quart., vol.1, no.1.

KRISHNAMOORTHY, M. (1969), "A multiple Selection Procedure in Minimaximal Path of a Disjunctive Graph", Presented in Second Annual Convention of OR Society of India, Bombay.

LAWLER, E.L. (1964), "On Scheduling Problems with Deferral Costs", Man. Sci., vol.11, no.2.

LAWLER, E.L. (1973), "Optimal Sequencing of a Single Machine Subject to Precedence Constraints", Man. Sci., vol.19, no.5.

LAWLER, E.L. and MOORE, J.M. (1969), "A Functional Equation and Its Application to Resource Allocation and Sequencing Problems", Man. Sci., vol. 16.

LOMNICKI, Z.A. (1965), "A Branch and Bound Algorithm for the Exact Solution of the Three Machine Scheduling Problem", Opn. Res. Quart., vol. 16, no.1.

LOMNICKI, Z.A. and BROWN, AP.G. (1966) - See BROWN, A.P.G. and LOMNICKI, Z.A. (1966).

Mc MAHON, G.B. and BURTON, P.G. (1967), "Flow Shop Scheduling with the Branch and Bound Method", Opn. Res., vol.15, no.3.

Mc NAUGHTON, R. (1959), "Scheduling with Dead Lines and Loss Functions", Man. Sci., vol. 6, no.1.

MANNE, A.S. (1960), "On the Job-Shop Scheduling Problem", Opn. Res., vol. 8, no.2.

MAXWELL, W.L., CONWAY, R.W. and MILLER, L.W. (1967), Theory of Scheduling, Addison-Wesley, Reading, Mass.

MOORE, J.M. (1968), "An n Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs", Man. Sci., vol.15, no.1.

NABESHIMA, I. (1967), "On the Bound of Makespans and Its Application in M Machine Scheduling Problem", J. Jap. Opn. Res. Soc., vol.9, nos. 3 and 4.

NABESHIMA, I. (1971), "General Scheduling Algorithm with Application to Parallel Scheduling and Multiproject Scheduling", J. Jap. Opn. Res. Soc., Vol. 14.

NUGENT, C.E. (1964), "On Sampling Approaches to the Solution of the n-by-m Static Sequencing Problem", Ph.D. Thesis, Cornell University.

RAMAMURTHY, K.G. (1973), "A Note on Two Stage Sequencing Problem with Two Parallel Machines at the First Stage", unpublished note.

RAMAMURTHY, K.G. and ARTHANARI, T.S. (1971) - See ARTHANARI, T.S. and RAMAMURTHY, K.G.

RAO, C.R. (1952), "Advanced Statistical Methods in Biometric Research, Wiley, New York.

RAU, J.G. (1971), "Minimising a Function of Permutations of n Integers", Opn. Res., vol.19, no.1.

ROOT, J.G. (1965), "Scheduling with Dead Lines and Loss Functions on k Parallel Machines", Man. Sci., vol. 11, no.3.

ROTHKOPF, M.H. (1966), "Scheduling Independent Tasks on Parallel Processors", Man. Sci., vol. 12, no.5.

RUSPINI, E.H. (1969), "A new Approach to Clustering", Information and Control, vol. 15, no.1.

SCHRAGE, L. (1970), "Solving Resource-Constrained Network Problems by Implicit Enumeration - NonPreemptive Case", Opn. Res., vol.18.

SHWIMER, J. (1972), "On the N-Job One-Machine Sequencing-Independent Scheduling Problem with Tardiness Penalties : A Branch and Bound Solution", Man. Sci., vol.18, no.6.

SMITH, W.E. (1956), "Various Optimizers for Single-Stage Production", Nav. Res. Log. Quart., vol.3, no.1.

SMITH, R.D. and DUDEK, R.A. (1967), "A General Algorithm for Solution of the n-Job, M-Machine Sequencing Problem of the Flow Shop", Opn. Res., vol. 15, no.1.

SRINIVASAN, V. (1971), "A Hybrid Algorithm for the One Machine Sequencing Problem to Minimize Total Tardiness", Nav. Res. Log. Quart., vol. 18, no.3.

SZWARC, W. (1968), "On Some Sequencing Problems", Nav. Res. Log. Quart., vol.15, no.2.

SZWARC, W. (1971), "Elimination Methods in the m x n Sequencing Problem", Nav. Res. Log. Quart., vol.18, no.3.

SZWARC, W. (1973), "Optimal Elimination Methods in the m x n Flow Shop Problems, Opn. Res., vol.21, no.6.

VINOD, H.D. (1969), "Integer Programming and the Theory of Grouping", J.A.S.A., vol.64.

WAGNER, H.M. (1959), "An Integer Linear-Programming Model for Machine Scheduling", Nav. Res. Log. Quart., vol.6, no.2.

WAGNER, H.M. and STORY, A.E. (1963), "Computational Experience with Integer Programming for Job-Shop Scheduling", Industrial Scheduling, J.F. Muth and G.C. Thompson, eds. Englewood Cliffs, N.J. : Prentice-Hall.

WAGNER, H.M. and GIGLIO, R.J. (1964), "Approximate Solutions to the Three-Machine Scheduling Problem", Opn. Res., vol.12, no.2.

WARD, J.H. (1963), "Hierarchical Grouping to Optimize an Objective Function", J.A.S.A., vol.58.

**********&*******