STUDIES
ON
MACHINE SCHEDULING PROBLEMS

(機械スケジューリング問題の研究)

TERUO MASUDA

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Machine Scheduling Problems

Machine scheduling problems originally arise from industrial production systems. In the system, we must perform a number of *jobs* by using a number of *machines*. Each of the jobs consists of many *operations*. To perform a job, we must process each of its operations. The *processing* of an operation requires the use of a particular machine during a particular duration, the *processing time* of operation. In these situations, a possible solution corresponds to a processing order of jobs on each machine. The goodness of obtained solution is measured by total time or total cost function reflecting actual purposes. The object of this thesis is to develop efficient algorithms giving the most preferable solutions in such actual problems.

The scheduling problems also occur under many other circumstances. In these circumstances, however, above terminologies are given more flexible interpretations: jobs and machines can stand for patients and hospital equipments, classes and teachers, ships and dockyards, programs and computers or cities and travelling salesmen. Each of these situations fits into the framework sketched above and thus falls within the scope of machine scheduling theory. Moreover, in most situations suggested above, if we choose poor sequencing decisions, we are sure to incur intolerably long times or large costs. Therefore we need to develop an efficient method (algorithm) for finding an optimal or at least sufficiently near optimal schedules of the jobs with respect to the given cost function (objective function).

Usually, the schedules are represented visually. The most popular visual representation is Gantt chart or timing diagram illustrated in Fig.1.1. In the figure, it is obvious that one of horizontal lines except for the topmost line corresponds to a machine and the topmost line represents time axis. The hatched areas in the figure represent idle periods on the machines. In this way, the Gantt chart is convenient to give an informal but intuitive notion of a schedule. More formally speaking, a schedule is defined as a suitable mapping that assigns a sequence of one or more disjoint execution intervals on each machine to each job without breaking the following restrictions.

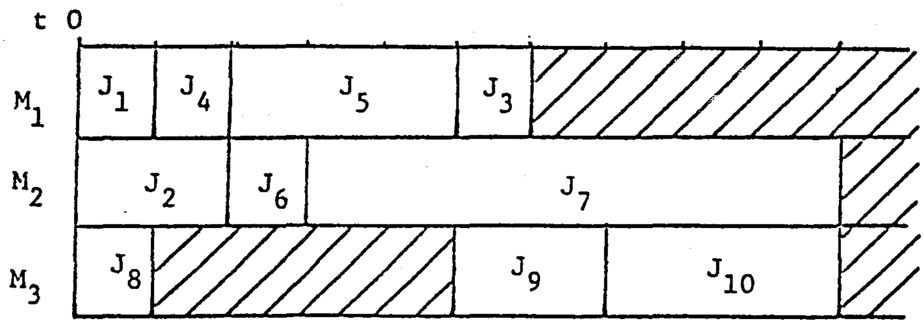


Fig.1.1. Example of Gantt chart.

(1) Each machine can process at most one job at the same time.

(2) Each job can be processed on at most one machine at the same time.

(3) The total length of the intervals assigned to the job is precisely its processing time.

(4) At least one machine is busy so long as there remains at least one uncompleted job.

(5) The jobs can be processed independently, that is, there exists no precedence constraint such that some job must be completed before other job can begin.

The above restrictions (1)-(5) are assumed throughout this thesis without being specially mentioned.

1.2 Classification of Machine Scheduling Problems

In the last section, we presented a general model of the machine scheduling problems. Thus, in a general setting of the machine scheduling problems, a set of jobs $J=\{J_1,\cdots,J_n\}$ has to be processed on a set of machines $M=\{M_1,\cdots,M_m\}$. Besides the general setting, the actual machine scheduling problems occurring under various circumstances have various characteristics. And, each problem can be specified principally by the characteristics of jobs, machines, and optimality criteria. Thus, we can classify the machine scheduling problems according to the above characteristics in the subsequent subsections.

1.2.1 Jobs

One of the most important characteristics of jobs is the number n of jobs to be processed. In this thesis, n is always assumed to be an arbitrary positive integer. Further, for each J_i, we should assign the following values.

(i) *The number of operations*. Each job J_i consists of m_i operations, each of which has to be processed on the machines with a particular function for the operation.

(ii) *Processing times*. To complete the processing of J_i, we must process each operation of J_i on a particular machine M_j depending on each operation during p_{ij} time in total. In particular, if p_{ij} does not depend on j, we denote it by p_i. Usually, the processing times are arbitrary positive constants. But sometimes we deal with the case that the processing times are all equal to the unit time, i.e., p_i or $p_{ij}=1$.

(iii) *Due dates*. The processing of each job should ideally be completed by the due dates. These due dates are denoted by

d_i or d_{ij}. But, we do not always impose the due dates for the jobs, depending on the objective.

1.2.2 Machines

One of the important characteristics of machines is the number m of available machines. Here, m is an arbitrary positive integer. Especially, the important special cases are the cases of two and three machines. Thus, given an m machine scheduling problem, we must develop a solution procedure that works effectively for any m.

Further, we have to classify the scheduling problems by the types of machines according to the difference in functional capability and speeds of machines. First, we classify the types of machines into shop type and parallel type. Moreover, we differentiate each of shop and parallel type machines into the particular types analyzed in the subsequent chapters.

(I) Shop Type Machines

In this type, each job J_i consists of m operations $O_{i1},\cdots,O_{im}$. Each operation O_{ij} can be processed only on M_j and can not be processed on any other machines. The processing time of O_{ij} is p_{ij}. Thus, each machine has the distinct functional capability. For example, in a computer system, an input device and an output device have clearly the different functional capability. The shop type machine is classified further into the following types by the order of processing of operations.

(i) Flow Shop Type Machines (F)

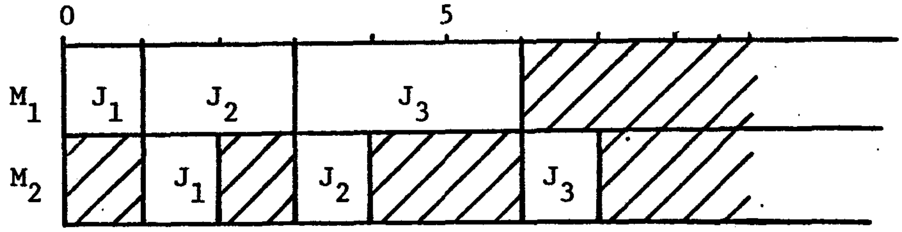
Each operation O_{ij} of J_i must complete processing on M_j before starting to process the next operation O_{ij+1} on M_{j+1} for $j=1,2,\cdots,m-1$. Thus, all jobs must pass through the machines in the same order, $M_1 \rightarrow M_2 \rightarrow \cdots \rightarrow M_m$. (See Fig. 1.2(a).)

(ii) Open Shop Type Machines (O)

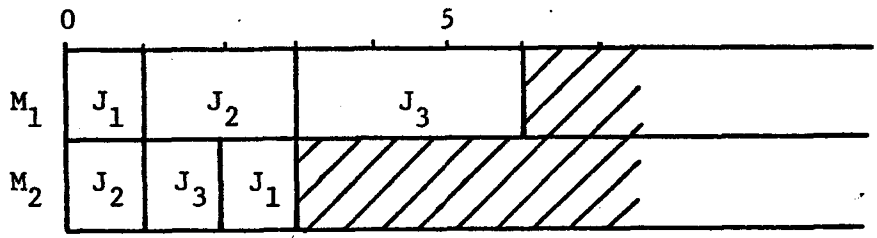
Each operation of J_i can pass through the machines in an arbitrary order, but more than one operations of J_i can not be processed at the same time. (See Fig. 1.2.(b).)

The above characteristics of machines are also the characteristics of jobs. Therefore we may call the jobs to be processed on the open and flow shop type machines the open shop type jobs and the flow shop type jobs, respectively.

The difference of the schedules on the above two shop type machines is illustrated in Fig. 1.2, where on both types processing times are taken as $p_{11}=1$, $p_{12}=1$, $p_{21}=2$, $p_{22}=1$, $p_{31}=3$ and $p_{32}=1$.



(a) A schedule on flow shop type machines.



(b) A schedule on open shop type machines.

Fig.1.2. The difference of the schedules on flow and open shop type machines.

In the above two shop types, we assume that the machines (jobs) have only the characteristic of either the flow shop type machines (jobs) or the open shop type machines (jobs). However, it is possible that the machines have the characteristics of both types simultaneously.

(iii) Mixed Shop Type Machines (MX)

The machines may have the characteristics of both the flow shop type machines and the open shop type machines simultaneously. In other words, in a set of jobs the flow shop type jobs and the open shop type jobs is mixed.

In the above three shop types, we assumed that the machines have same speeds. But in some cases, the speed of each machine can be changeable. Thus, the speed of each machine must be determined together with the schedule.

(iv) Generalized Mixed Shop Type Machines (GMX)

The jobs are the mixed shop type jobs. And, each speed of the machines is not a constant but a variable to be determined together with the schedule in the final solution. The actual processing time of operation O_{ij} of job J_i on machine M_j is $p_{ij} = p'_{ij}/s_j$, where s_j is a variable speed of M_j and p'_{ij} is an amount of processing requirement of J_i.

Next, we consider the parallel type machines.

(II) Parallel Type Machines

Each job consists of only one operation. By the speed of each machine, we differentiate the parallel type machines as the following. Here, each machine has the same function and each job can be processed on any machines.

(v) Identical Parallel Type Machines (I)

Each machine has the same functional capability and same speed, and each job can be processed on any machines. The processing time p_{ij} of job J_i on machine M_j is equal to p_i for $j=1,2,\cdots,m$.

(vi) Uniform Parallel Type Machines (U)

Each job can be processed on any machines. Each machine has the same functional capability, but its speed is different and fixed. Thus, the processing time p_{ij} of each job J_i on machine M_j is $p_{ij}=p_i/q_j$, where q_j is the predetermined speed of M_j and p_i the processing requirement of J_i.

The former is identical in both functional capability and speed. Thus, we may regard the machines as the m identical machines. On the other hand, the latter is identical in functional capability, but each machine has a different constant speed. Thus, some machine can process the jobs faster(or slower) than other machines.

In the following, we extend the case of constant speeds to the variable speeds. In this case, similar to the generalized open shop case, each speed is to be determined together with the schedule.

(vii) Generalized Uniform Parallel Type Machines (GU)

In this type, the speed of each machine is not constant but variable. Therefore we must determine the speed of each machine together with the schedule. The processing time p_{ij} of job J_i on machine M_j is $p_{ij}=p_i/s_j$, where s_j is the changeable speed of M_j and p_i is a processing requirement. The other characteristics are same to those of the uniform parallel type.

In the above three parallel types, we assume that each machine can be processed on any machine. In the following type, we remove that assumption. Thus, job J_i can not be always processed on any machine but can be processed only on a predetermined subset Q_i of M.

(viii) Quasi-Identical Parallel Type Machines (QI)

Each job J_i can be processed only on a predetermined subset Q_i of machine set M. For example, let $M=\{M_1, M_2, M_3\}$ and $Q_1=\{M_1, M_3\}$. Then, job J_1 can be processed on M_1 and M_3, and can not be processed on M_2. Therefore the processing times of job J_i on machine M_j are $p_{ij}=p_i$ if $M_j \in Q_i$ and $p_{ij}=\infty$ if $M_j \notin Q_i$. The other characteristics are the same as those of the identical parallel type.

1.2.3 Optimality criteria

In the above subsections, we pointed out the characteristics of jobs and machines to classify the scheduling problems. The remaining factor is the optimality criterion. In this subsection, we define the optimality criteria to be chosen. First, we define the following quantities for each job J_i.

(a) Completion time; the time C_i at which the processings of all the operations of job J_i complete, namely $C_i = \max_{1 \leq j \leq m} (C_j(i))$, where $C_j(i)$ is the completion time of the processing of operation O_{ij}.

(b) Lateness; the difference L_i between the completion time and the due dates of job J_i, namely $L_i = \max_{1 \leq j \leq m} (L_{ij})$, where $L_{ij}=C_i-d_{ij}$ is the lateness of job J_i on M_j.

Using the above quantities, we define the optimality criteria.

(i) Minimizing Maximum Completion Time (C_{max})

The optimality criterion is to minimize the maximum completion time $C_{max} = \max_{1 \leq i \leq n} C_i$. In other words, we want to complete all the jobs as soon as possible.

(ii) Minimizing Maximum Lateness (L_{max})

The objective is to minimize maximum lateness $L_{max} = \max_{1 \leq i \leq n} L_i$. Here, we want to complete each job before the due dates as soon as possible.

(iii) Minimizing General Cost Function (f_{max})

The optimality criterion is to minimize the sum of the cost concerning the maximum completion time and the cost incurred by changing the machine speeds.

1.2.4 Representation of models

To represent symbolically each scheduling problem, we introduce 4-tuple notation $\alpha|\beta|\gamma|\delta$. The first letter α shows the number n of jobs to be processed, where n is an arbitrary positive integer.

The second letter β is the number m of machines, where m is an arbitrary positive integer.

The third letter γ specifies the machine types. We use the notations as the following table.

γ	machine types
F	flow shop
O	open shop
MX	mixed shop
GMX	generalized mixed shop
I	identical parallel
U	uniform parallel
GU	generalized uniform parallel
QI	quasi-identical parallel

Table 1.1. The notations specifying the machine types.

The last letter δ is the optimality criterion, as follows.

δ	objective
C_{max}	maximum completion time
L_{max}	maximum lateness
f_{max}	general cost function

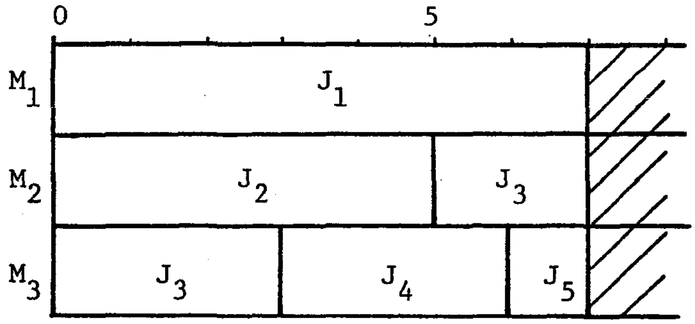
Table 1.2. Optimality criteria.

Example 1.1. $n|2|F|L_{max}$: minimize maximum lateness of n jobs on two flow shop type machines.

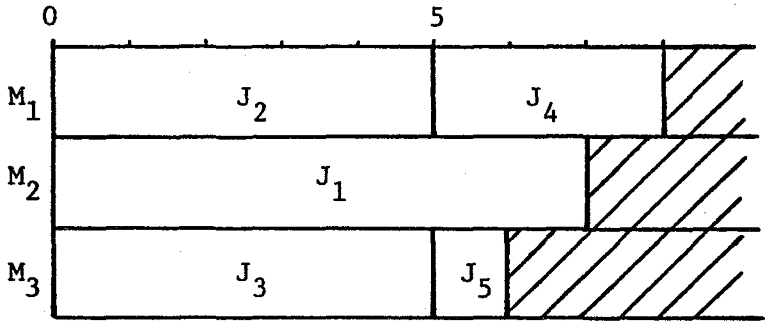
$n|2|MX|C_{max}$: minimize maximum completion time of n jobs on two mixed shop type machines.

$n|m|I|C_{max}$: minimize maximum completion time of n jobs on m identical parallel type machines.

Besides this notation, we may use the terminology, "preemptive" or "nonpreemptive" scheduling. In a preemptive schedule, the processing of any operation may be interrupted and resumed later again. The difference between the preemptive and nonpreemptive scheduling of five jobs on three identical machines is illustrated in Fig. 1.3, where $p_1=7$, $p_2=5$, $p_3=5$, $p_4=3$, and $p_5=1$.



(a) Preemptive schedule.



(b) Nonpreemptive schedule

Fig.1.3. Examples for preemptive and nonpreemptive schedules.

1.3 Computational Complexity

In this section, we shall review briefly the theory of computational complexity, especially the NP-completeness theory, because most of the machine scheduling problems fall into the class of NP-complete problems.

Historically, the theory of computability initiated by A. Turing [26] played an important role to stimulate attention to the existence of problems in which no algorithm can solve. The following problem is one of the most well-known such undecidable problems.

<u>Halting Problem of Programs</u>: Given an arbitrary computer program and an arbitrary input to that program, can we decide whether or not the program will eventually halt when applied to that input?

A variety of other problems are now known to be undecidable, including Hilbert's tenth problem [22] and several problems of tiling the plane [2].

Note here that an algorithm is said to solve a problem if it gives a solution within finite steps for any instance of the problem.

<u>*The complexity of Algorithm*</u>: In general, any mathematical problem can be described in terms of some parameters and free variables. An instance of the problem is obtained by given particular values for problem parameters. For the purpose of computerization, the discrete free variables as well as the parameters are assumed to be *binary encoded* in a *string* which becomes an input of the computer. The length of the string is called the *input length* (or the *size of the instance*). The complexity

of an algorithm is called O(g(n)), if its running time is always bounded by a certain function $c \cdot g(n)$, where c is a constant, for all the values of the input length n. An *O(g(n)) time algorithm* is called *polynomial time algorithm*, if g(n) is a polynomial of n. The fundamental nature of the distinction between polynomial time algorithms and non-polynomial time algorithms has been discussed by J. Edmonds [3] and others [1], [4]. The polynomial time algorithms are known to be efficient algorithms by a rule of thumb. Therefore a problem with a polynomial time algorithm is called *tractable*, while a problem with no polynomial time algorithm is called *intractable*. The theory of NP-completeness provides a way to determine whether or not a given problem is intractable .

Nondeterministic Computation: We assume an ordinary sequential computer appended with the following fictitious instruction,

$$\mathrm{CHOICE}(L_L, L_2, \cdots, L_k).$$

When the computer reads this instruction, it jumps to k instructions with labels $L_1, \cdots, L_k$ and executes them simultaneously. This may be considered as a model of parallel computations. However, in ordinary parallel computation models the number of instructions to be executed in parallel is fixed beforehand, while in the above computation model, each time a CHOICE instruction is encoutered, the computation path branches unlimitedly. Such computation is called *nondeterministic computation*.

Polynomial Reducibility: If the input data of problem A can be transformed into the input data of another problem B in polynomial time with respect to the input length of A, and if the solvability of A is equivalent to that of B, then A is said to be *polynomially reducible* to B.

NP-Completeness: Let class *NP* be the class of all the problems which can be solved within the time bounded by a polynomial function, if the nondeterministic computation is allowed. Similarly, let class *P* be the class of all the problems solved by deterministic polynomial time algorithms. It is clear that P⊆NP. The equality P=NP is considered to be highly unlikely for the following reason. P≠NP, however, has not yet proved.

A problem is said to be NP-complete if it is in the class NP and all the problems in NP is polynomially reducible to it. See references [1] and [4]. The NP-complete problems are the hardest problems in NP in the sense that if any one of them were to have a polynomial time algorithm, then all the problems in NP will do so. This shows that P=NP if and only if one of the NP-complete problems has a polynomial time algorithm. So far, thousands of problems have been proved to be NP-complete, and about 300 among them are listed in [4]. The fact that no polynomial time algorithm has been found for them is a strong circumstantial evidence that P≠NP.

1.4 Coping with NP-complete Problems

Proving the NP-completeness of a given problem is only the starting point of the analysis of the problem but never is the terminal point. It is easy to show that most of the machine scheduling problems we encounter in the real world are NP-complete. In many situations, however, it may be sufficient to obtain some good but not optimal solutions. In this section, we mention some directions to cope with NP-complete problems. They are of both practical and theoretical importance, and have been intensively studied recently.

Investigation of Some Solvable Cases of Problems by Imposing Restrictions: Even if a given problem is NP-complete, it may contain some cases of practical importance which can be solved easily. In Section 2.3, we will introduce such an example for $n|3|O|C_{max}$ nonpreemptive scheduling problem.

Development of Approximation Algorithms and Their Worst Case Bounds: In many situations, not optimal but good solutions are accepted by practitioners. Thus, it is very important to develop some approximation algorithms efficiently providing approximately good solutions. Further, to evaluate the effectiveness of various approximation algorithms, we have to give their error bounds for the worst case (*worst case bounds*). These will be treated in Sections 2.2 and 3.2.

1.5 Outline of the Thesis

This thesis consists of four chapters. Chapters 2 and 3 are devoted to the conventional scheduling problems in which all machines have the same predetermined machine speeds, while Chapter 4 deals with the scheduling problems in which each machine speed is a variable.

Chapter 2 discusses the scheduling problems on shop type machines. First, we study an $n|2|F|L_{max}$ nonpreemptive scheduling problem. Since the problem is already known to be NP-complete, we present a solvable case and propose an approximation algorithm. Further, the worst case bound is obtained. Second, this chapter deals with a solvable case for $n|3|O|C_{max}$ nonpreemptive scheduling problem. Finally, we develop a polynomial time algorithm constructing an optimal schedule of $n|2|MX|C_{max}$ nonpreemptive scheduling problem.

Chapter 3 discusses the three scheduling problems on parallel type machines. First, we consider an $n|m|I|L_{max}$ nonpreemptive scheduling problem. This problem is again NP-complete. Therefore we propose two approximation algorithms, one of which is based on the earliest due date rule and the other is its refinement. And, the worst case bounds for each of them are derived. Second, we deals with an $n|2|I|L_{max}$ preemptive scheduling problem with generalized due dates. For this problem, we develop a polynomial time algorithm to minimize maximum lateness. Third, this chapter deals with $n|m|QI|C_{max}$ nonpreemptive and preemptive scheduling problems. Since the former is NP-complete, we give a solvable case in which job J_i has a unit processing time. For the latter, we develop a polynomial time algorithm constructing an optimal schedule.

Chapter 4 is devoted to extending the ordinary scheduling problems with constant machine speeds to the ones with changeable machine speeds. First, we discuss an $n|m|GU|f_{max}$ preemptive scheduling problem. This problem is an extension of $n|m|U|C_{max}$ preemptive scheduling problem to the case with variable speeds. Polynomial algorithms are presented to find optimal speed assignments for a variety of cost functions. Further, we show that if we relax some of assumptions for this problem, the resulting problems become NP-hard. Second, we deal with an $n|2|GMX|f_{max}$ nonpreemptive scheduling problem, which is an extension of $n|2|MX|C_{max}$ nonpreemptive scheduling problem to the case with variable speeds. For this problem, similarly, we develop a polynomial algorithm to find an optimal speed assignment.

CHAPTER 2

SCHEDULING PROBLEMS ON SHOP TYPE MACHINES

2.1 Introduction

In this chapter, we discuss scheduling problems on shop type machines. When the objective is to minimize the maximum completion time for two machines in shop, the problems were solved already as shown below.

Shop	m	Complexity	Reference
flow shop	2	$O(n \log n)$	[15]
open shop	2	$O(n)$	[5]

Johnson's procedure is known as Johnson's rule. Though there is no advantage for the preemption in these two machine cases, in the case of more than two machines, the restriction of nonpreemption makes the problem NP-complete for both shops [9]. On the other hand, in the preemptive case, Gonzalez and Sahni again developed optimal algorithms [5].

In section 2.2, we consider a nonpreemtive scheduling problem on the two machine flow shop whose objective is to minimize the maximum lateness. (Abbreviated to $n|2|F|L_{max}$ nonpreemptive scheduling problem according to Section 1.2.) Since this problem becomes NP-complete, we first present a solvable case where the relation between due dates and processing times is restricted. Next, we propose an approximation algorithm based on Johnson's rule which constructs an optimal schedule for $n|2|F|C_{max}$ scheduling problem, and give its worst case bound.

In Section 2.3, we discuss a nonpreemptive scheduling problem to minimize the maximum completion time on three machine open shop, i.e., $n|3|O|C_{max}$ nonpreemptive scheduling problem. This problem also becomes NP-complete. Therefore we present a solvable case which has two kinds of jobs. In this case, each job J_i has a zero processing time on at least one of M_2 and M_3, i.e., $p_{i2}=0$ or $p_{i3}=0$.

In Section 2.4, we deal with a nonpreemptive scheduling problem minimizing the maximum completion time on two machine mixed shop, namely $n|2|MX|C_{max}$ nonpreemptive scheduling problem. For this problem, we develop a polynomial time algorithm giving an optimal schedule.

For the simplicity of notations, throughout this chapter, we use a_i, b_i and c_i in place of p_{i1}, p_{i2} and p_{i3}, respectively, as processing times of operations O_{i1}, O_{i2} and O_{i3} of job J_i. Further, we assume that machine speeds are the same for all machines.

2.2 Solvable Case and Some Bound on Approximation Algorithm for $n|2|F|L_{max}$ Nonpreemptive Scheduling Problem

The problem dealt with is described as follows; (i) a set of n jobs $J=\{J_1,\cdots,J_n\}$ is to be processed on two machines M_1 and M_2, (ii) each job J_i has the two processing times a_i and b_i corresponding to M_1 and M_2, (iii) due dates of job J_i are the same for both machines, i.e., $d_{i1}=d_{i2}=d_i$, (iv) the processing of J_i must complete on M_1 before starting to process on M_2, (v) the objective is to minimize the maximum lateness.

We assume that $d_1 \leq d_2 \leq \cdots \leq d_n$.

For the maximum lateness problem on a single machine, Jackson [13] has obtained an exact algorithm which finds an optimum schedule in a polynomial time of problem size. Furthermore, Lawler [17] has obtained $O(n^2)$ exact algorithm for the related problem with arbitrary nondecreasing cost function and general precedence constraints.

With respect to scheduling problems with due dates, however, very few worst case bounds have been obtained. (See Graham et al. [9] for details.) Kise et al. have developed effective approximation algorithms and showed their worst case bounds for the maximum lateness problem on a single machine. In general, to evaluate the effectiveness of approximation algorithms, various measures such as the absolute deviation $\omega-\omega'(\Pi)$ and the relative deviation $(\omega-\omega'(\Pi))/\omega$ have been customarily used so far, where ω denotes the value for the objective under cosideration for optimal schedule and $\omega'(\Pi)$ the value for approximate schedule generated by the approximation algorithm Π. As pointed out by Kise et al. however, above measures exhibit a shortcoming that they give different values for two equivalent problems, where equivalence means that one problem is obtained by applying a simple transformation to the other, and the optimal and the approximate schedules are the same in both problems. This pathology urges us to employ the modified relative deviation,

$$\frac{\omega-\omega'(\Pi)}{\omega+d_{max}},$$

proposed by Kise et al. as an effective measure of approximation algorithm Π, where $d_{max}=\max\{d_i \mid i=1,2,\cdots,n\}$.

In the sequel, we first present a solvable case in the sense that the optimal schedule can be found easily. Then we propose an approximation algorithm for general $n|2|F|L_{max}$ nonpreemptive scheduling problem and obtain its modified relative deviation or its worst case bound.

2.2.1 Solvable case for $n|2|F|L_{max}$ nonpreemptive scheduling problem

General $n|2|F|L_{max}$ nonpreemptive scheduling problem is NP-complete. Therefore, we first consider a solvable case in the sense that an optimal schedule can be found easily. We assume that for $1\leqq i,j\leqq n$,

(C) $d_i\leqq d_j \leftrightarrow \min(a_i,b_j,)\leqq\min(a_j,b_i)$.

EDD rule: *EDD rule* schedules jobs according to nondecreasing due dates, i.e., in the order, $J_1,J_2,\cdots,J_n$.

Theorem 2.1. If the assumption (C) holds, EDD rule constructs an optimal schedule for $n|2|F|L_{max}$ nonpreemptive scheduling problem.

Proof. The completion time C_i of job J_i scheduled by EDD rule is given by Johnson's formulation as follows;

$$C_i = \max_{1 \leq u \leq i} \{ \sum_{j=1}^{u} a_j + \sum_{j=u}^{i} b_j \}$$

$$= \max\{ C_{i-1}, \sum_{j=1}^{i} a_j \} + b_i .$$

(See [15].) Then, the lateness L_i of job J_i becomes as follows;

$$L_i = C_i - d_i$$

$$= \max \{ C_{i-1}, \sum_{j=1}^{i} a_j \} + b_i - d_i .$$

Similarly,

$$L_{i+1} = C_{i+1} - d_{i+1}$$

$$= \max\{ C_i, \sum_{j=1}^{i+1} a_j \} + b_{i+1} - d_{i+1}$$

$$= \max\{ \max(C_{i-1}, \sum_{j=1}^{i} a_j) + b_i, \sum_{j=i}^{i+1} a_j \} + b_{i+1} - d_{i+1}$$

$$= \max\{ C_{i-1} + b_i, \sum_{j=1}^{i} a_j + b_i, \sum_{j=1}^{i+1} a_j \} + b_{i+1} - d_{i+1} .$$

Let L_i' and L_{i+1}' be the latenesses of i-th and (i+1)-th jobs in the schedule obtained by interchanging jobs J_i and J_{i+1}. (In the resulting schedule, i-th job is J_{i+1} and (i+1)-th job is J_i.) In other words, L_i' is the lateness of job J_{i+1} in the resulting schedule and L_{i+1}' that of job J_i. Thus we have

$$L_i' = \max\{ C_{i-1}, \sum_{j=1}^{i-1} a_j + a_{i+1} \} + b_{i+1} - d_{i+1} ,$$

$$L'_{i+1}=\max\{C_{i-1}+b_{i+1},\ \sum_{j=1}^{i-1}a_j+a_{i+1}+b_{i+1},\ \sum_{j=1}^{i+1}a_j\}+b_i-d_i.$$

First, we show that

$$\max(L_i,\ L_{i+1})\leq\max(L'_i,\ L'_{i+1}).$$

Since $d_i\leq d_{i+1}$ and $\min(a_i,\ b_{i+1})\leq\min(a_{i+1},\ b_i)$, we have $L_i\leq L'_{i+1}$ and $L'_i\leq L_{i+1}$. Therefore, it is enough to prove $L_{i+1}\leq L'_{i+1}$.

Case (i) $a_i\leq b_{i+1}$.

Note that inequalities $a_i\leq a_{i+1}$ and $a_i\leq b_i$ also hold in this case.

Subcase (i-a) $L_{i+1}=C_{i-1}+b_i+b_{i+1}-d_{i+1}$.

From $d_i\leq d_{i+1}$, we have

$$L_{i+1}=C_{i-1}+b_i+b_{i+1}-d_{i+1}\leq C_{i-1}+b_i+b_{i+1}-d_i\leq L'_{i+1}.$$

(i-b) $L_{i+1}=\sum_{j=1}^{i}a_j+b_i+b_{i+1}-d_{i+1}$.

Since $d_i\leq d_{i+1}$ and $a_i\leq a_{i+1}$, we prove

$$\begin{aligned}L_{i+1}&=\sum_{j=1}^{i}a_j+b_i+b_{i+1}-d_{i+1}\\&\leq\sum_{j=1}^{i-1}a_j+a_{i+1}+b_i+b_{i+1}-d_i\\&\leq L'_{i+1}.\end{aligned}$$

(i-c) $L_{i+1}=\sum_{j=1}^{i+1}a_j+b_{i+1}-d_{i+1}$.

By $d_i\leq d_{i+1}$ and $a_i\leq b_i$, we have

$$\begin{aligned}L_{i+1}&=\sum_{j=1}^{i+1}a_j+b_{i+1}-d_{i+1}\\&\leq\sum_{j=1}^{i-1}a_j+a_{i+1}+b_i+b_{i+1}-d_i\leq L'_{i+1}\end{aligned}$$

Thus if $a_i \leqq b_{i+1}$, then we have $L_{i+1} \leqq L'_{i+1}$.

Case (ii) $b_{i+1} < a_i$.

Note that $b_{i+1} \leqq a_{i+1}$ and $b_{i+1} \leqq b_i$ also hold in this case. We can prove $L_{i+1} \leqq L'_{i+1}$ by the similar manner to Case (i). Therefore if $\min(a_i, b_{i+1}) \leqq \min(a_{i+1}, b_i)$ and $d_i \leqq d_{i+1}$, then $\max(L_i, L_{i+1}) \leqq \max(L'_i, L'_{i+1})$.

Let C'_k and L'_k be the completion time and the lateness of job J_k in the schedule obtained by interchanging jobs J_i and J_{i+1}. For $k<i$, it is clear that $C'_k = C_k$ and $L'_k = L_k$. Since $C'_k \geqq C_k$ holds for $k>i+1$ by virtue of Johnson's rule, we have $L'_k \geqq L_k$.

Thus since the relation (C) holds among all jobs and is transitive, we prove the theorem by repeating the pairwise interchanges of adjacent jobs. □

The problem under consideration is NP-complete, so it seems likely that an efficient algorithm does not exist. Therefore enumerative methods such as branch-and-bound ones may be the only available methods for obtaining optimal solution.

One may suspect that we can decrease the number of enumerations by applying Theorem 2.1 to a number of job pairs for some of which the relation (C) holds. The following example shows the case that the conjecture fails.

Example 2.1. Let $J=\{J_1, J_2, J_3\}$,

$a_1=2$, $b_1=5$, $d_1=55$,

$a_2=10$, $b_2=1$, $d_2=50$,

$a_3=4$, $b_3=100$ and $d_3=60$.

In this example, though $\min(a_1, b_3) \leqq \min(a_3, b_1)$ and $d_1 \leqq d_3$, the optimal schedule is given in Fig. 2.1. The maximum lateness in the optimal schedule is $L^*_{max} = 55$.

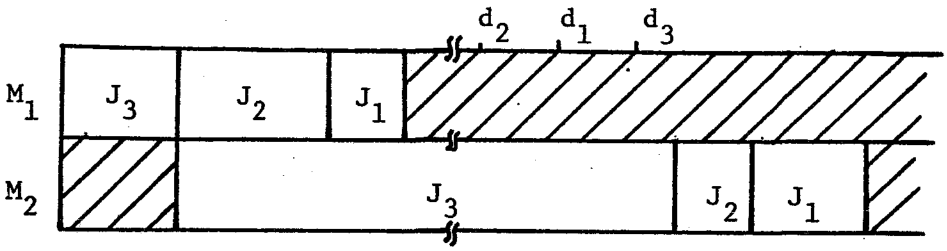


Fig. 2.1. An optimal schedule of Example 2.1.

2.2.2 Bound on approximation algorithm for $n|2|F|L_{max}$ nonpreemptive scheduling problem

In subsection 2.2.1, we showed a solvable case of $n|2|F|L_{max}$ scheduling problem. Unfortunately, general problem is NP-complete. Therefore in this subsection, we give an approximation algorithm and show how it behaves in the worst case. We call the algorithm based on EDD rule algorithm FEDD, which assigns the jobs according to EDD rule for flow shop type machines. We first prove Lemma 2.1 giving the bound of the maximum completion time when a set of jobs is scheduled by algorithm FEDD.

Lemma 2.1. Let C' be the maximum completion time of schedule induced by algorithm FEDD and C* that of schedule constructed by Johnson's rule. (See [15].) Then we have

$$\frac{C'}{C^*} \leq 2.$$

Proof. It is clear that

$$C^* \geq \max\left(\sum_{i=1}^{n} a_i,\ \sum_{i=1}^{n} b_i\right).$$

Also it follows that

$$C' \leqq \sum_{i=1}^{n} (a_i+b_i) \leqq 2\cdot\max(\sum_{i=1}^{n} a_i, \sum_{i=1}^{n} b_i) \leqq 2C^*.$$

Thus we prove

$$\frac{C'}{C^*} \leqq 2. \quad \square$$

Next, we show that without loss of generality we may assume that job J_n with a maximum due date determines the maximum lateness of algorithm FEDD.

Lemma 2.2. For certain number K, let $\bar{J}=\{\bar{J}_1,\cdots,\bar{J}_n\}$ be the minimal job set for which modified relative deviation of FEDD exceeds K, i.e.,

$$\frac{L(\bar{J};FEDD)-L(\bar{J};\Pi^*)}{L(\bar{J};\Pi^*)+d_n} > K$$

holds. Then, $\bar{J}_n$ determines the maximum lateness of FEDD, $L(\bar{J};FEDD)$.

Proof. We prove this lemma by contradiction. We assume that job $\bar{J}_i$, $i<n$, determines the maximum lateness of algorithm FEDD. Let $J'=\{\bar{J}_1,\bar{J}_2,\cdots,\bar{J}_n\}$ be the subset of $\bar{J}$ obtained by eliminating jobs $\bar{J}_{i+1},\cdots,\bar{J}_n$ from $\bar{J}$. Clearly

$$L(\bar{J};FEDD)=L(J';FEDD),$$

$$L(\bar{J};\Pi^*) \geqq L(J';\Pi^*),$$

and

$$d_n = \max_{1\leqq j\leqq n} d_j \geqq \max_{1\leqq j\leqq i} d_j = d_i$$

hold. These imply

$$K < \frac{L(\bar{J};FEDD)-L(\bar{J};\Pi^*)}{L(\bar{J};\Pi^*)+d_n} \leqq \frac{L(J';FEDD)-L(J';\Pi^*)}{L(J';\Pi^*)+d_i}.$$

Consequently, we have a smaller job set J'. This contradicts the minimality of job set $\bar{J}$. Thus, job $\bar{J}_n$ determines the maximum lateness of algorithm FEDD. $\square$

Using these lemmas, we obtain a bound on algorithm FEDD.

Theorem 2.2. Let L'_{max} and L^*_{max} be the maximum latenesses of the schedules constructed by applying algorithm FEDD and any optimal algorithm for an $n|2|F|L_{max}$ nonpreemptive scheduling problem, respectively. Then we have

$$\frac{L'_{max} - L^*_{max}}{L^*_{max} + d_n} \leqq 1.$$

Further, this bound is asymptotically the best possible.

Proof. Since the first half of this proof will be proved by contradiction, it is sufficient to develop a relationship only for the smallest n for which the theorem may be violated. Thus, we assume that a job set J defines a minimal job set for which the theorem does not hold.

Now by Lemma 2.2, we may consider only the case $L'_{max}=C'-d_n$, where C' is the same as defined in Lemma 2.1. It is clear that

$$L^*_{max} \geqq C^*-d_n.$$

Thus

$$\frac{L'_{max}-L^*_{max}}{L^*_{max}+d_n} \leqq \frac{C'-d_n-(C^*-d_n)}{C^*} = \frac{C'}{C^*} -1.$$

Since $2 \geqq C'/C^*$ by Lemma 2.1, we prove

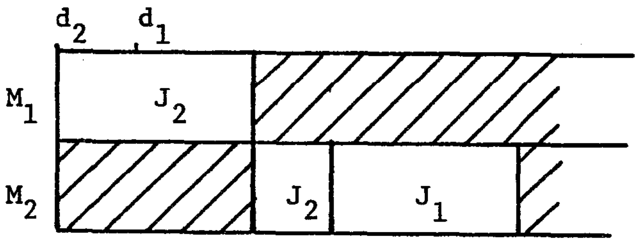
$$\frac{L'_{max} - L^*_{max}}{L^*_{max}+d_n} \leq 1.$$

The following example shows that this bound is asymptotically the best possible.

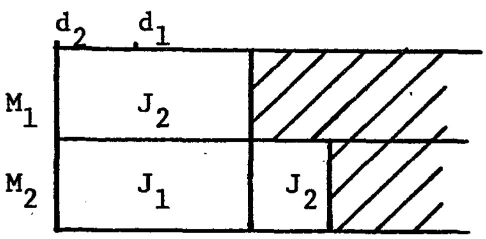
Let $a_1=0$, $b_1=K$, $d_1=\varepsilon(>0)$, $a_2=K$, $b_2=\varepsilon$ and $d_2=0$, where K is an arbitrary positive constant. For this instance, the approximate and the optimal schedules are given in Fig. 2.2(a) and (b), respectively. Then, we have $L'_{max}=2K+\varepsilon-\varepsilon=2K$ and $L^*_{max}=K+\varepsilon-0=K+\varepsilon$. Therefore, we prove

$$\frac{L'_{max} - L^*_{max}}{L^*_{max}+d_n} = \frac{K-\varepsilon}{K+2\varepsilon} \longrightarrow 1 \ (\varepsilon \to 0).$$

This completes the proof of the theorem. ▯



(a) Approximate schedule.



(b) Optimal schedule

Fig. 2.2. An asymptotically tight example for Theorem 2.2.

2.3 A Solvable Case for $n|3|O|C_{max}$ Nonpreemptive Scheduling Problem

In this section, we consider a set of jobs $J=\{J_1,\cdots,J_n\}$ to be processed on three machine open shop. Each job J_i consists of three operations, which have processing times $a_i>0$, $b_i\geqq0$ and $c_i\geqq0$, respectively. Each job J_i can pass through the machines M_1, M_2 and M_3 in an arbitrary order, but more than one operations of job J_i can not be processed simultaneously. Further, each job J_i must be processed nonpreemptively on any machines. Our objective is to minimize the maximum completion time. In general, this problem also becomes NP-complete [9]. Therefore we present a solvable case for $n|3|O|C_{max}$ nonpreemptive scheduling problem. To give a solvable case, we shall make the following assumptions.

Assumptions

(a) Let

$O_1=\{J_i\in O|c_i=0\}$,

$O_2=\{J_i\in O|b_i=0,\ c_i\neq0\}$,

and

$O=O_1\cup O_2$.

(b) Let J_q and J_r be the jobs such that

$b_q\geqq\max_{j\in O_1'}\{a_j\}$,

and

$c_r\geqq\max_{j\in O_2'}\{a_j\}$,

where $O_1'=\{J_i\in O_1|a_i<b_i\}$ and $O_2'=\{J_i\in O_2|a_i<c_i\}$. (Note that J_q belongs to O_1 and J_r to O_2.)

If $\sum_{J_i\in O_1} b_i \geqq \max(\sum_{J_i\in O_1} a_i, \max_{J_i\in O_1}(a_i+b_i))$, then we assume that $a_q+b_q \leqq \sum_{J_i\in O_1} a_i$. Similarly, if $\sum_{J_i\in O_2} c_i \geqq \max(\sum_{J_i\in O_2} a_i, \max_{J_i\in O_2}(a_i+c_i))$, then $a_r+c_r \leqq \sum_{J_i\in O_2} a_i$.

By the assumption (a), if either O_1 or O_2 is empty, this solvable case reduces to $n|2|O|C_{max}$ scheduling problem. For $n|2|O|C_{max}$ scheduling problem, Gonzalez and Sahni developed an $O(n)$ time algorithm constructing an optimal schedule. The forms of optimal schedules generated by Gozalez and Sahni algorithm (G-S algorithm) are classified into the six types in Fig. 2.2, if we ignore the processing order on each machine. On the schedules of types I and I', machine M_2 may have an idle period though there exist uncompleted jobs, but machine M_1 has no idle period as long as there exist uncompleted jobs. On the other hand, concerning types II , II', III and III',there exists no idle period on M_1 and M_2 except for the first and the last time periods. (Note that if we remove the assumption (b), on types II and II' M_1 may have an idle period other than the first or the last interval.)

In the next subsection, we specify the starting times of jobs based on the solution of $n|2|O|C_{max}$ scheduling problem rather than the processing order of jobs on each machine.

2.3.1 Construction of optimal schedule

We present a construction method of optimal schedule under the assumptions (a) and (b). To determine any schedule, it is sufficient to specify either the processing order or the starting times of jobs on each machine. In this subsection, we specify the starting times of jobs on each machine. Now, if either subset O_1 or O_2 is empty, the problem reduces to $n|2|O|C_{max}$ scheduling

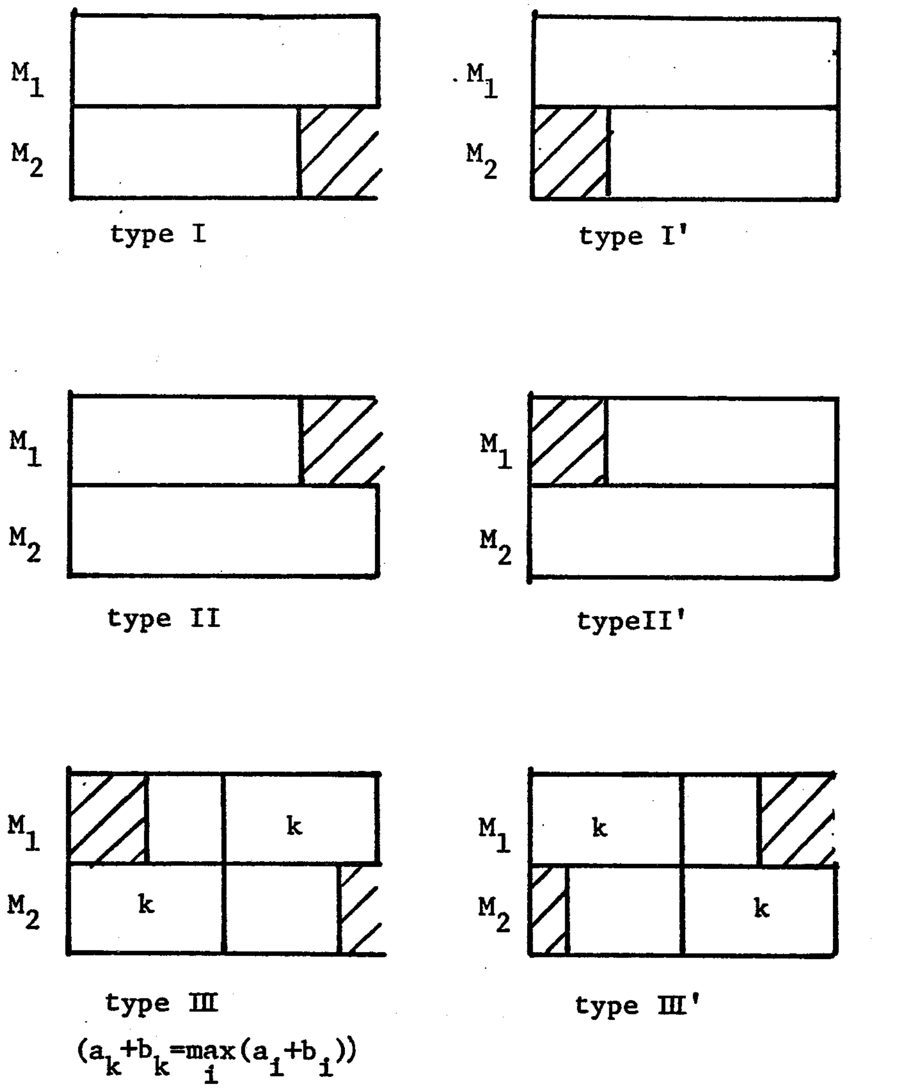


Fig. 2.2. The forms of optimal schedules for $n|2|O|C_{max}$ scheduling problem.

problem. So let F_1^* (F_2^*) be a value of optimal schedule constructed by applying G-S algorithm for the jobs in O_1 (O_2). Further, let $s_j'(i)$ be the starting time of job J_i ($\in O$) on machine M_j in the schedules constructed by G-S algorithm. By Gonzalez and Sahni [5], the possible values of F_1^* and F_2^* are max (A_1, B, a_k+b_k) and max(A_2, C, $a_{k'}+c_{k'}$), where $A_1 = \sum_{J_i \in O_1} a_i$, $B = \sum_{J_i \in O_1} b_i$, $a_k+b_k = \max_{J_i \in O_1} (a_i+b_i)$, $A_2 = \sum_{J_i \in O_2} a_i$, $C = \sum_{J_i \in O_2} c_i$ and $a_{k'}+c_{k'} = \max_{J_i \in O_2} (a_i+c_i)$. Then, the combinations of F_1^* and F_2^* are only the following nine pairs.

	F_1^*	F_2^*
(1)	A_1	A_2
(2)	B	A_2
(3)	a_k+b_k	A_2
(4)	A_1	C
(5)	B	C
(6)	a_k+b_k	C
(7)	A_1	$a_{k'}+c_{k'}$
(8)	B	$a_{k'}+c_{k'}$
(9)	a_k+b_k	$a_{k'}+c_{k'}$

However, since (4), (7) and (6) are reducible to (2), (3) and (8), respectively, by the appropriate exchange of notations, e.g., M_2 and M_3, we can focus on the following six cases.

Case	F_1^*	F_2^*
1	A_1	A_2
2	B	A_2
3	a_k+b_k	A_2

Case	F_1^*	F_2^*
4	B	C
5	B	$a_{k'}+c_{k'}$
6	a_k+b_k	$a_{k'}+c_{k'}$

For these six cases, we present the starting time, $s_j(i)$, of job J_i on machine M_j in our solvable case.

Case 1. $F_1^*=A_1$ and $F_2^*=A_2$

For this case, G-S algorithm generates a schedule of either type I or I' for both O_1 and O_2. The schedule of type I(I'), however, can be transformed to type I'(I) by reversing the processing order of jobs on each machine, since on open shop type machines any job can pass through machines in an arbitrary order. Therefore without loss of generality we may assume that the optimal schedules for O_1 and O_2 are the schedules of type I and I', respectively. Then we present the starting times of job J_i on machines M_1, M_2 and M_3 as follows.

$$s_1(i)=\begin{cases} s_1'(i) & \text{for } J_i\epsilon O_1 \\ s_1'(i)+A_1 & \text{for } J_i\epsilon O_2 \end{cases}$$

$$s_2(i)=\begin{cases} s_2'(i) & \text{for } J_i\epsilon O_1 \\ 0 & \text{for } J_i\epsilon O_2 \end{cases}$$

$$s_3(i)=\begin{cases} 0 & \text{for } J_i\epsilon O_1 \\ s_3'(i)+A_1 & \text{for } J_i\epsilon O_2 \end{cases}$$

In this case, the constructed schedule is the one illustrated in Fig. 2.3.

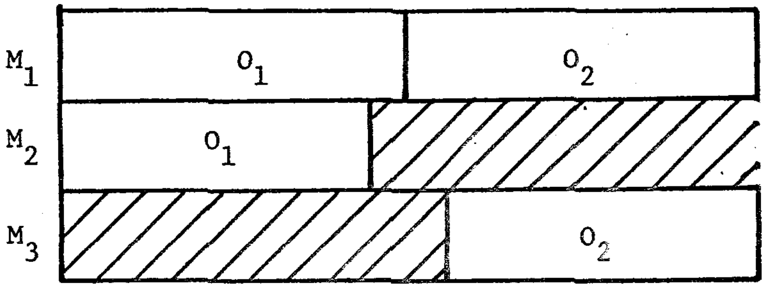


Fig. 2.3. The typical schedule for Case 1.

<u>Case 2.</u> $F_1^*=B$ and $F_2^*=A_2$

For this case, we may assume that the schedule for O_1 is type II and that for O_2 is type I'. Then we define the starting times of each job J_i on machines M_1, M_2 and M_3 as follows.

$$s_j(i)=s_j'(i) \qquad \text{for } j=1,2 \text{ and } J_i\in O_1$$

$$s_j(i)=s_j'(i)+\max(A_1,\ B-A_2) \qquad \text{for } j=1,3 \text{ and } J_i\in O_2$$

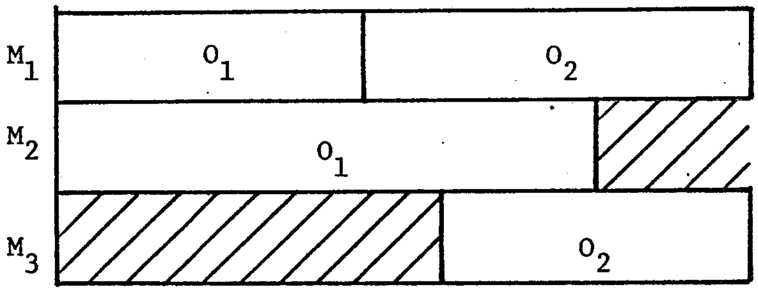
The typical schedules characterizing this case are illustrated in Fig. 2.4.

<u>Case 3.</u> $F_1^*=a_k+b_k$ and $F_2^*=A_2$

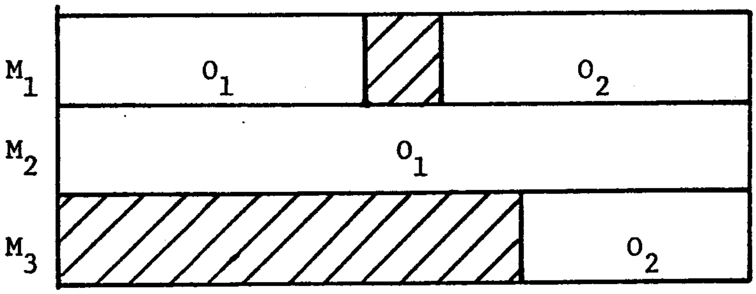
We have the schedule of type III' for the job in O_1 and of type I' for the job in O_2. Here, we define the starting times as follows.

$$s_j(i)=s_j'(i) \qquad \text{for } j=1,2 \text{ and } J_i\in O_1$$

$$s_j(i)=s_j'(i)+\max(A_1\ ,\ a_k+b_k-A_2) \qquad \text{for } j=1,3 \text{ and } J_i\in O_2$$



(a) $B \leqq A_1+A_2$.



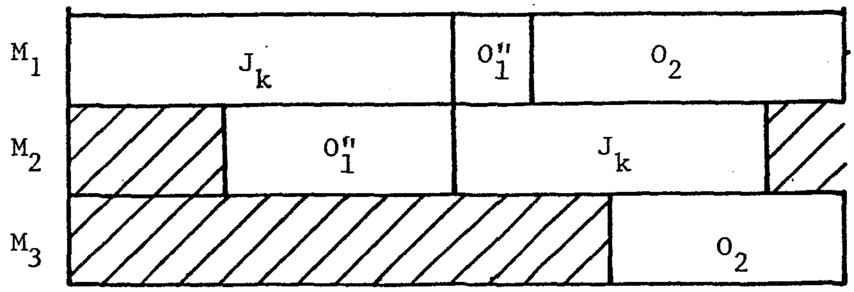
(b) $B > A_1+A_2$.

Fig. 2.4. The typical schedules for Case 2.

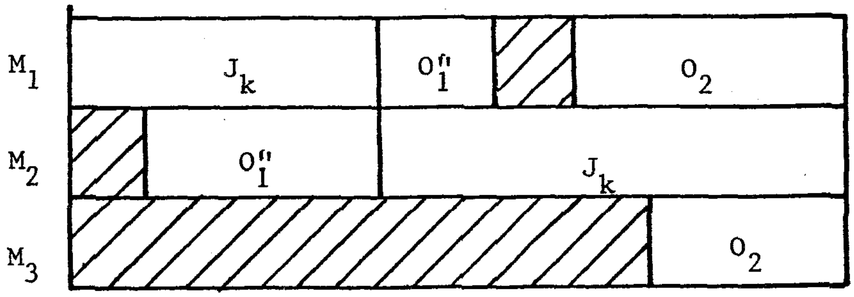
The representative schedules for this case are illustrated in Fig.2.5.

<u>Case 4.</u> $F_1^*=B$ and $F_2^*=C$

In this case, let the schedule for O_1 be type II and for O_2 type II'. Then we set the starting times for each job J_i on machines M_1, M_2 and M_3 as follows.



(a) $a_k+b_k \leqq A_1+A_2$. $(O_1''=O_1-\{J_k\})$



(b) $a_k+b_k > A_1+A_2$.

Fig. 2.5. The typical schedules for Case 3.

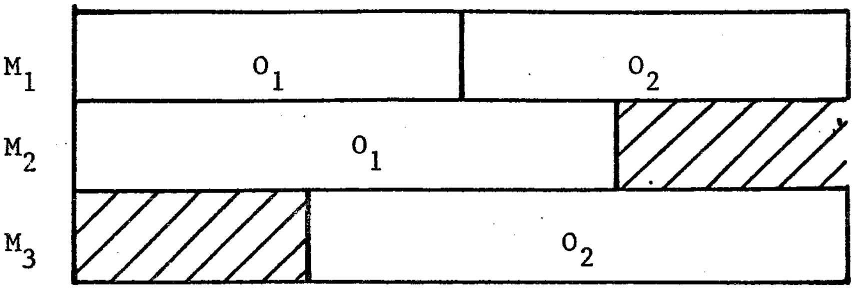
$$s_j(i)=s_j'(i) \quad \text{for } j=1,2 \text{ and } J_i \in O_1$$

$$s_j(i)=s_j'(i)+\max(A_1+A_2-C,\ 0,\ B-C) \quad \text{for } j=1,3 \text{ and } J_i \in O_2$$

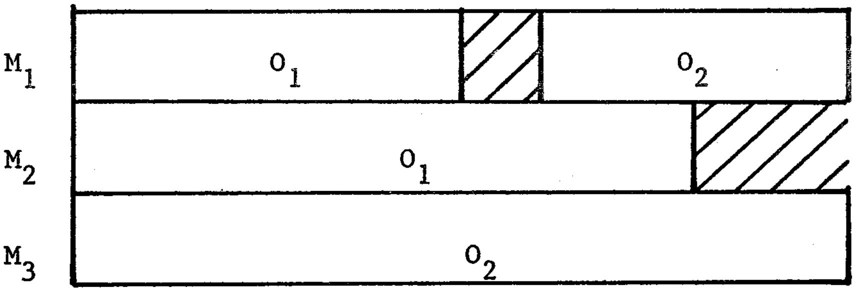
We have three typical schedules illustrated in Fig. 2.6 for this case.

Case 5. $F_1^*=B$ and $F_2^*=a_{k'}+c_{k'}$

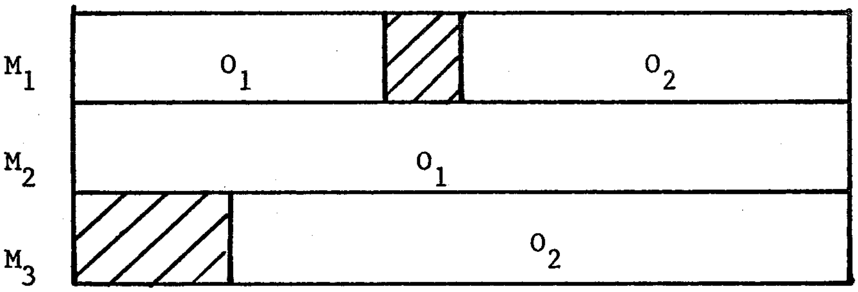
For this case, the schedules for the jobs in O_1 and O_2 are



(a) $A_1+A_2 \geqq \max(B, C)$.



(b) $C \geqq \max(A_1+A_2, B)$.



(c) $B > \max(A_1+A_2, C)$.

Fig. 2.6. The typical schedules for Case 4.

those of type II and type III, respectively. We set the starting times for each job J_i as follows.

$$s_j(i)=s_j'(i) \quad \text{for } j=1,2 \text{ and } J_i \in O_1$$

$$s_j(i)=s_j'(i)+\max(B-(a_{k'}+c_{k'}),\ A_1+A_2-(a_{k'}+c_{k'}),\ 0)$$
$$\text{for } j=1,3 \text{ and } J_i \in O_2$$

Those typical schedules characterizing this case are illustrated in Fig. 2.7.

<u>Case 6.</u> $F_1^*=a_k+b_k$ and $F_2^*=a_{k'}+c_{k'}$

For this case, the types of schedules for the jobs in O_1 and O_2 are type III' and type III, respectively. Then the starting times for job J_i are set as

$$s_j(i)=s_j'(i) \quad \text{for } j=1,2 \text{ snd } J_i \in O_1$$

and

$$s_j(i)=s_j'(i)+\max(A_1+A_2-(a_{k'}+c_{k'}),\ (a_k+b_k)-(a_{k'}+c_{k'}),\ 0)$$
$$\text{for } j=1,3 \text{ and } J_i \in O_2.$$

The typical schedules for this case are illustrated in Fig. 2.8.

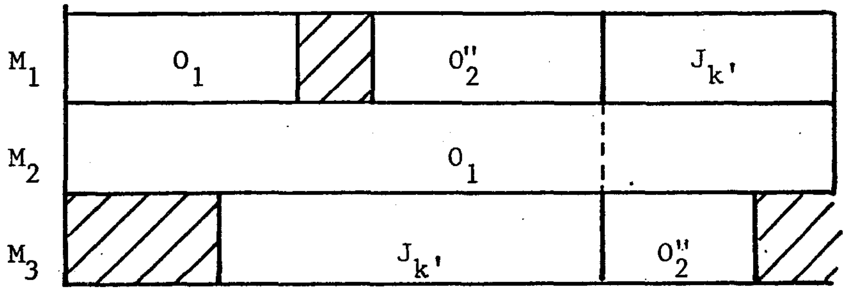
In the next subsection, we prove the validity of the starting times given in this subsection, such that the schedule based on the above starting times becomes an optimal schedule.

2.3.2 Proof of validity

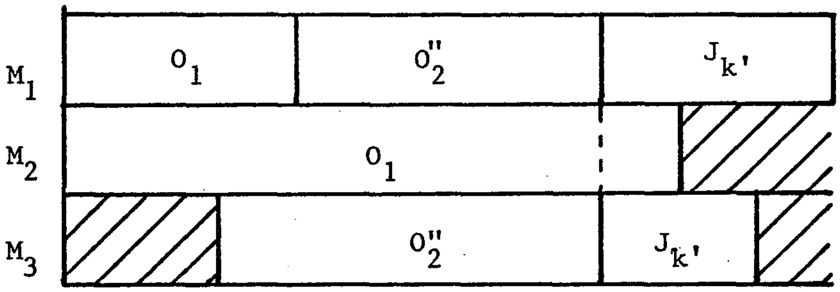
We prove the validity of the starting times presented in the last subsection.

If either O_1 or O_2 is empty, then the validity is trivial. So we assume that both O_1 and O_2 are nonempty.

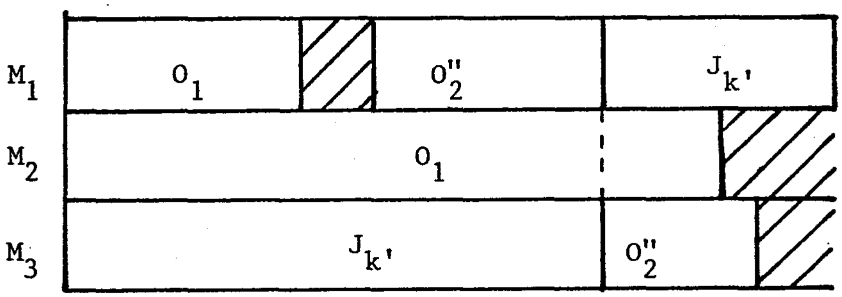
To prove the validity, we must show that the constructed



(a) $B \geqq \max(A_1+A_2,\ a_{k'}+c_{k'})$ $(O_2''=O_2-\{J_{k'}\}.)$

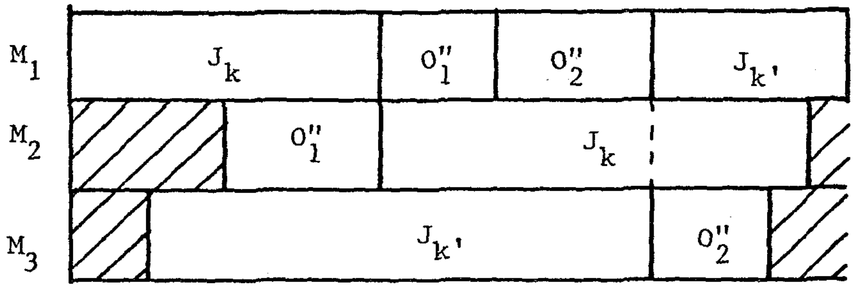


(b) $A_1+A_2 \geqq \max(B,\ a_{k'}+c_{k'})$.

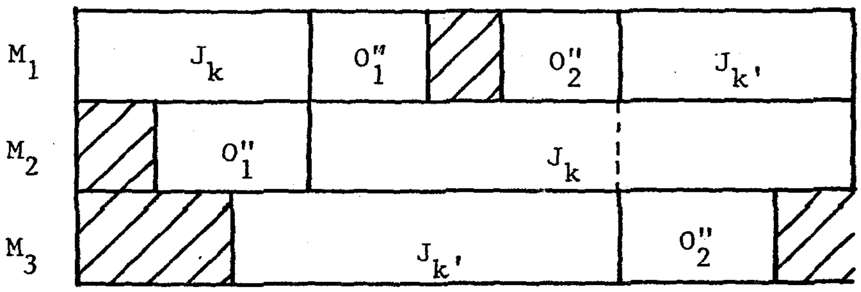


(c) $a_{k'}+c_{k'} > \max(A_1+A_2,\ B)$.

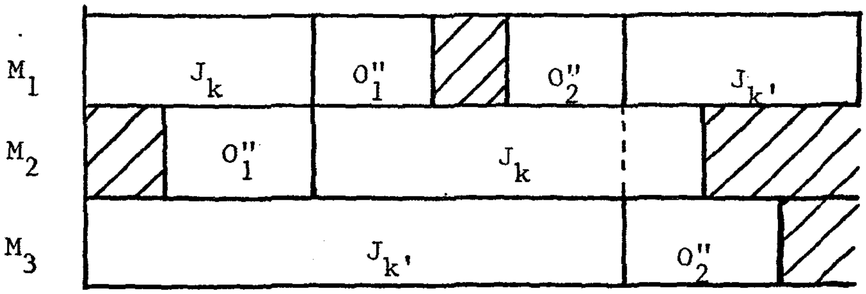
Fig. 2.7. The typical schedule for Case 5.



(a) $A_1+A_2 \geqq \max(a_k+b_k,\ a_{k'}+c_{k'})$. $(O_1''=O_1-\{J_k\},\ O_2''=O_2-\{J_{k'}\}.)$



(b) $a_k+b_k \geqq \max(A_1+A_2,\ a_{k'}+c_{k'})$.



(c) $a_{k'}+c_{k'} > \max(A_1+A_2,\ a_k+b_k)$.

Fig. 2.8. The typical schedules for Case 6.

schedule is feasible, i.e., $C_1(i) \leqq s_2(i)$ or $C_2(i) \leqq s_1(i)$ for $J_i \in O_1$ and $C_1(i) \leqq s_3(i)$ or $C_3(i) \leqq s_2(i)$ for $J_i \in O_2$, where $C_j(i)$ is the completion time of J_i on M_j, and the value of that schedule is the minimum of the completion times.

(1) Feasibility.

To show the feasibility of the schedule, it is sufficient to show that for job J_i in O_1, either

$$C_1(i) \leqq s_2(i) \text{ or}$$

$$C_2(i) \leqq s_1(i)$$

holds, and for job J_i in O_2, either

$$C_1(i) \leqq s_3(i) \text{ or}$$

$$C_3(i) \leqq s_1(i)$$

holds.

Now in the schedules generated by G-S algorithm, we have

$$(2.1) \qquad \begin{cases} C_1'(i) \leqq s_2'(i) \text{ or} \\ C_2'(i) \leqq s_1'(i) \end{cases} \quad \text{for } J_i \in O_1$$

and

$$(2.2) \qquad \begin{cases} C_1'(i) \leqq s_3'(i) \text{ or} \\ C_3'(i) \leqq s_1'(i) \end{cases} \quad \text{for } J_i \in O_2$$

where $C_1'(i)$, $C_2'(i)$ and $C_3'(i)$ are the completion times of those schedules for job J_i on M_1, M_2 and M_3, respectively. From the setting of start times, we have

$$(2.3) \qquad \begin{cases} s_j(i) = s_j'(i) \text{ and} \\ C_j(i) = C_j'(i) \end{cases} \quad \text{for } j=1,2 \text{ and } J_i \in O_1$$

and

$$(2.4)\quad \begin{cases} s_j(i)=s'_j(i)+K \\ C_j(i)=C'_j(i)+K \end{cases} \quad \text{for } j=1,3 \text{ and } J_i \in O_2,$$

where $K= \begin{cases} A_1 \quad \text{for Case 1,} \\ \max(A_1,\ B-A_2) \quad \text{for Case 2,} \\ \max(A_1,\ a_k+b_k-A_2) \quad \text{for Case 3,} \\ \max(A_1+A_2-C,\ 0,\ B-C) \quad \text{for Case 4,} \\ \max(B-(a_{k'}+c_{k'}),\ A_1+A_2-(a_{k'}+c_{k'}),\ 0) \quad \text{for Case 5,} \\ \max(A_1+A_2-(a_{k'}+c_{k'}),\ (a_k+b_k)-(a_{k'}+c_{k'}),\ 0) \quad \text{for Case 6.} \end{cases}$

Consequently, by substituting (2.3) and (2.4) into (2.1) and (2.2), respectively, we can prove the feasibility.

Optimality.

Here, we prove the optimality of schedule based on the starting times set in the last subsection, i.e., that the schedule has the minimum value of maximum completion time, C^*_{max}. Gonzalez and Sahni showed the lower bound of C^*_{max}, LB, for $n|m|O|C_{max}$ scheduling problem as follows.

$$C^*_{max} \geqq LB=\max(\max_i \sum_{j=1}^{m} p_{ij},\quad \max_j \sum_{i=1}^{n} p_{ij})$$

where p_{ij} is the processing time of operation O_{ij}. For our solvable case, this lower bound is reduced to

$$LB=\max(A_1+A_2,\ B,\ C,\ a_k+b_k,\ a_{k'}+c_{k'}).$$

Further, corresponding to each case, this lower bound is rewritten as,

$$LB=\begin{cases} A_1+A_2 & \text{for Case 1,} \\ \max(A_1+A_2,\ B) & \text{for Case 2,} \\ \max(A_1+A_2,\ a_k+b_k) & \text{for Case 3,} \\ \max(A_1+A_2,\ B,\ C) & \text{for Case 4} \\ \max(A_1+A_2,\ B,\ a_{k'}+c_{k'}) & \text{for Case 5,} \\ \max(A_1+A_2,\ a_k+b_k,\ a_{k'}+c_{k'}) & \text{for Case6.} \end{cases}$$

In the sequel, we will prove that this lower bound is achievable in all cases. Let C_1, C_2 and C_3 be the maximum completion times on machines M_1, M_2 and M_3.

Since, as we assumed, both O_1 and O_2 are nonempty, for $J_i \in O_1$ and $J_{i'} \in O_2$ we have $s_1(i)+a_i \leq s_1(i')$ in all cases. Then, the maximum completion times on the machines are

$$(2.5)\quad \begin{cases} C_1 = \max\limits_{J_i \in O_2} (s_1(i)+a_i) \\ C_2 = \max\limits_{J_i \in O_1} (s_2(i)+b_i) \\ C_3 = \max\limits_{J_i \in O_2} (s_3(i)+c_i) \end{cases}$$

Also, let C_1' and C_3' be the maximum completion times of the schedule generated by G-S algorithm for O_2 on M_1 and M_3, respectively, and C_2' be that for O_1 on M_2.

Substituting (2.3) and (2.4) into (2.5), we have

$$(2.6)\quad \begin{cases} C_1 = \max\limits_{J_i \in O_2} (s_1'(i)+a_i)+K = C_1'+K \\ C_2 = \max\limits_{J_i \in O_1} (s_2'(i)+b_i) = C_2' \\ C_3 = \max\limits_{J_i \in O_2} (s_3'(i)+c_i)+K = C_3'+K. \end{cases}$$

Thus, from (2.6) we have

$$\text{(2.7)} \qquad \begin{aligned} C_{max} &= \max(C_1, C_2, C_3) \\ &= \max(C_1'+K, C_2', C_3'+K) \\ &= \max(\max(C_1', C_3')+K, C_2'). \end{aligned}$$

Now since in our construction $C_2' \leq A_1 < C_1'+K$ if $F_1^* = A_1$ and $C_2' = \max(B, a_k+b_k)$ if $F_1^* \neq A_1$, and since $\max(C_1', C_3') = F_2^*$, we have

$$C_{max} = \max(F_2^*+K, B, a_k+b_k).$$

Further, since in each case it holds that

$$\begin{aligned}
&A_1 \geqq \max(a_k+b_k, B), && \text{if } F_1^*=A_1, \\
&B \geqq \max(a_k+b_k, A_1), && \text{if } F_1^*=B, \\
&a_k+b_k \geqq \max(A_1, B), && \text{if } F_1^*=a_k+b_k, \\
&A_2 \geqq \max(a_{k'}+c_{k'}, C), && \text{if } F_2^*=A_2, \\
&C \geqq \max(A_2, a_{k'}+c_{k'}), && \text{if } F_2^*=C, \\
&a_{k'}+c_{k'} \geqq \max(A_2, C), && \text{if } F_2^*=a_{k'}+c_{k'},
\end{aligned}$$

these inequalities and the definition of K together show that

$$C_{max} = \begin{cases} A_1+A_2 & \text{for Case 1,} \\ \max(A_1+A_2, B) & \text{for Case 2,} \\ \max(A_1+A_2, a_k+b_k) & \text{for Case 3,} \\ \max(A_1+A_2, B, C) & \text{for Case 4,} \\ \max(A_1+A_2, B, a_{k'}+c_{k'}) & \text{for Case 5,} \\ \max(A_1+A_2, a_k+b_k, a_{k'}+c_{k'}) & \text{for Case 6.} \end{cases}$$

Therefore we have proved that C_{max} equals the lower bound in all cases.

2.4 The Mixed Shop Scheduling Problem

We consider a set of jobs $J=\{1,2,\cdots,n\}$[†] to be processed nonpreemptively on two machine mixed shop. Job i has processing times $a_i \geqq 0$ and $b_i \geqq 0$ on machines M_1 and M_2, respectively. The job set J consists of two disjoint subsets F and O, i.e., $J=F \cup O$ and $F \cap O=\phi$. Each job i in F must complete the processing of operation O_{i1} on M_1 before starting to process O_{i2} on M_2, i.e., F is a set of flow shop type jobs. On the other hand, O is a set of open shop type jobs. Thus each job i in O must complete the processing on M_k before starting to process on M_j, where $j,k=1,2$ and $j \neq k$.

Let $C_j(i)$ be the completion time of job i, and $S_j(i)$ be the starting time of job i on machine M_j for $j=1,2$. Further, let $|F|=n_1$, $|O|=n_2$ and $n=n_1+n_2$. Then, for any $i \in F$, $C_1(i) \leqq S_2(i)$ must hold and for any $i \in O$ either $C_1(i) \leqq S_2(i)$ or $C_2(i) \leqq S_1(i)$ must hold. The schedule is nonpreemptive. The objective is to find the schedule minimizing the maximum completion tims $\max_{i \in J}(C_1(i), C_2(i))$. (Abbreviated to $n|2|MX|C_{max}$ nonpreemptive scheduling problem.) When O is empty, this problem is reduced to the two machine flow shop problem solved by Johnson [15]. In this special case, the solution procedure obtaining the optimal schedule is known as Johnson's rule; if $\min(a_i,b_j) \leqq \min(a_j,b_i)$, then the processing of job i precedes the processing of job j. The case which has only open shop type jobs has been solved by Gonzalez and Sahni [5]. Further, Jackson [14] has solved the two machine

---

[†] In this section, we use job "i" instead of job "J_i" for the simplicity of notation.

scheduling problem with two distinct job sets such that the processing order of jobs must be M_1 to M_2 or M_2 to M_1. In our problem, the flexibility of processing order for some jobs, however, is taken into account, which is more complicated than that of Jackson. An extension of the algorithm to Jackson type of mixed shop nevertheless would be interesting. Also it seems unlikely to solve our problem by straightforward extension of the Jackson's method.

2.4.1 Preliminaries

We prove some lemmas needed for the proof of optimality of algorithms.

Let $A_F=\sum_{i\in F} a_i$, $B_F=\sum_{i\in F} b_i$, $A_O=\sum_{i\in O} a_i$ and $B_O=\sum_{i\in O} b_i$. Further, let $LF=(f_1,f_2,\cdots,f_{n_1})$ be the list such that the jobs in F are ordered according to Johnson's rule, i.e., for $1\leqq i\leqq j\leqq n_1$, $\min(a_{f_i}, b_{f_j})\leqq\min(a_{f_j},b_{f_i})$. For job f_i, CF^*, $C_1'(i)$ and $S_2'(i)$ are the maximum completion time, the completion time on M_1 and the starting time on M_2 in the schedule constructed by ordinary Johnson's procedure, respectively. Let I_i be the idle time between adjacent job pair f_{i-1} and f_i, i.e., $I_i=S_2'(i)-S_2'(i-1)-b_{f_{i-1}}$ for $1\leqq i\leqq n_1$, where $S_2'(0)=0$ and $I_1=a_{f_1}$. Then

$$C_1'(i)=\sum_{j=1}^{i} a_{f_j},$$

$$S_2'(i)=\sum_{j=1}^{i-1} b_{f_j}+\sum_{j=1}^{i} I_j,$$

and

$$CF^*=\sum_{j=1}^{n_1} b_{f_j}+\sum_{j=1}^{n_1} I_j=B_F+\sum_{j=1}^{n_1} I_j.$$

Lemma 2.3. The following inequalities hold for each f_i.

$$C_1'(i) \leqq S_2'(i) \leqq CF^*-B_F+\sum_{j=1}^{i-1} b_{f_j}.$$

Proof. By virtue of the ordinary Johnson's procedure, it is clear that $C_1'(i) \leqq S_2'(i)$ for each f_i. Since

$$CF^*-B_F+\sum_{j=1}^{i-1} b_{f_j} = \sum_{j=1}^{i-1} b_{f_j} + \sum_{j=1}^{n_1} I_j$$

and

$$I_j \geqq 0,$$

we can prove that

$$CF^*-B_F+\sum_{j=1}^{i-1} b_{f_j} \geqq \sum_{j=1}^{i-1} b_{f_j} + \sum_{j=1}^{i} I_j = S_2'(i). \qquad \square$$

Now, we define job sets O_1 and O_2 as follows.

$$O_1 = \{i \in O \mid a_i \geqq b_i\},$$

$$O_2 = \{i \in O \mid a_i < b_i\}.$$

Further, we choose r and l to be any two distinct jobs in O such that

$$b_r \geqq \max_{j \in O_2} (a_j),$$

$$a_l \geqq \max_{j \in O_1} (b_j).$$

Then let $LO=(s_1, s_2, \cdots, s_{n_2})$ be the list of jobs in O such that

$$s_1 = l, \quad s_{n_2} = r,$$

$s_j \in O_1 - \{l, r\}$ for $2 \leqq j \leqq k$ and $a_{s_{j-1}} \geqq a_{s_j}$ for $3 \leqq j \leqq k$,

$s_j \in O_2 - \{l, r\}$ for $k+1 \leqq j \leqq n_2-1$ and $b_{s_{j-1}} \leqq b_{s_j}$ for $k+2 \leqq j \leqq n_2-1$.

Lemma 2.4. If $A_0-a_r\leqq B_0-b_l$, then the following inequality holds.

$$\sum_{j=i+1}^{n_2} b_{s_j} \geqq \sum_{j=i}^{n_2-1} a_{s_j}.$$

Proof. For $i\leqq k$, we have

$$\sum_{j=i+1}^{n_2} b_{s_j}-\sum_{j=i}^{n_2-1} a_{s_j}=B_0-\sum_{j=1}^{i} b_{s_j}-\sum_{j=i}^{n_2-1} a_{s_j}$$

$$=B_0-b_{s_1}-(A_0-a_r)-\sum_{j=2}^{i} b_{s_j}+\sum_{j=1}^{i-1} a_{s_j}$$

$$=B_0-b_l-(A_0-a_r)+\sum_{j=1}^{i-1} (a_{s_j}-b_{s_{j+1}}).$$

Now, since $B_0-b_l\geqq A_0-a_r$, $a_{s_1}=a_l\geqq \max_{2\leqq j\leqq k} (b_{s_j})$ and $a_{s_j}\geqq a_{s_{j+1}}\geqq b_{s_{j+1}}$ for $2\leqq j\leqq k-1$, we can prove that

$$\sum_{j=i+1}^{n_2} b_{s_j} \geqq \sum_{j=i}^{n_2-1} a_{s_j}.$$

(Note that for $2\leqq j\leqq k$, $s_j\epsilon O_1$.) For $i\geqq k+1$, since $s_{n_2}=r$, $b_r\geqq \max_{k+1\leqq j\leqq n_2-1} (a_{s_j})$ and $b_{s_j}\geqq b_{s_{j-1}}>a_{s_{j-1}}$ for $k+2\leqq j\leqq n_2-1$, we have

$$\sum_{j=i+1}^{n_2} b_{s_j}-\sum_{j=i}^{n_2-1} a_{s_j}=\sum_{j=i+1}^{n_2} (b_{s_j}-a_{s_{j-1}})>0.$$ □

Lemma 2.5. If $A_0-a_r>B_0-b_l$, then we have

$$b_l+\sum_{j=1}^{i-1} a_{s_j}\geqq \sum_{j=1}^{i} b_{s_j}.$$

Proof. We can prove this lemma similar to Lemma 2.4 and so it is omitted. □

Lemma 2.6. Let C^*_{max} be the optimal value of the maximum completion time. Then the following inequality holds.

$$C^*_{max} \geqq \max(A_F+A_O,\ B_F+B_O,\ CF^*,\ \max_{i\in O}(a_i+b_i)).$$

Proof. Clearly the righthand side of the above inequality is a lower bound of C^*_{max}. []

2.4.2. Optimal algorithms

We give an exact algorithm for each of the following cases.

(1) $A_F \geqq B_O$,

(2) $A_F < B_O$ and $B_F \geqq A_O$,

(3) $A_F < B_O$ and $B_F < A_O$.

In the following, we use the same notations C_1 and C_2 to denote the maximum completion times on M_1 and M_2 of the schedule constructed by the algorithm given for each case. Further, let

$$C_{max}=\max(C_1,\ C_2),$$

i.e., C_{max} is the maximum completion time of that schedule.

Case 1: $A_F \geqq B_O$

We give the algorithm for the case of $A_F \geqq B_O$.

Algorithm I

(1) On M_1, process the jobs in F successively according to Johonson's rule from time 0. Then, process the jobs in O successively in an arbitrary order from time A_F after processing all flow shop type jobs.

(2) On M_2, process the jobs in O successively in an arbitrary order from time 0. Then, process the jobs in F according to Johnson's rule from time $\max(B_O,\ CF^*-B_F)$.

The typical cases of the schedule constructed by algorithm I are illustrated in Fig. 2.9.

Lemma 2.7. If $A_F \geqq B_O$, then algorithm I constructs an optimal schedule.

Proof. (i) Case $CF^*-B_F \geqq B_O$. For any $i \in O$, clearly

$$C_2(i) \leqq B_O$$

and

$$S_1(i) \geqq A_F$$

hold. From the above inequalities and the assumption $A_F \geqq B_O$, we have $C_2(i) \leqq S_1(i)$.

Since for any $f_i \in F$, $C_1(f_i)=\sum_{j=1}^{i} a_{f_j}$ and $S_2(f_i)=CF^*-BF+\sum_{j=1}^{i-1} b_{f_j}$,

by Lemma 2.3 we have $C_1(f_i) \leqq S_2(f_i)$. Further, the facts that the idle time on M_1 is zero and the idle time on M_2 is only the time interval between time B_O and CF^*-B_F show $C_1=A_F+A_O$ and $C_2=B_O+(CF^*-B_F-B_O)+B_F=CF^*$. Thus we obtain $C_{max}=\max(C_1, C_2)=\max(A_F+A_O, CF^*)$.

(ii) Case $CF^*-B_F<B_O$. For any $i \in O$, similar to the case (i) we can prove $C_2(i) \leqq S_1(i)$.

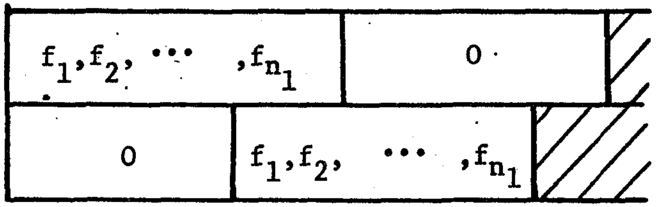
For any $f_i \in F$, we have $C_1(f_i)=\sum_{j=1}^{i} a_{f_j}$ and $S_2(f_i)=B_O+\sum_{j=1}^{i-1} b_{f_j}$.

Since $B_O<CF^*-B_F$,

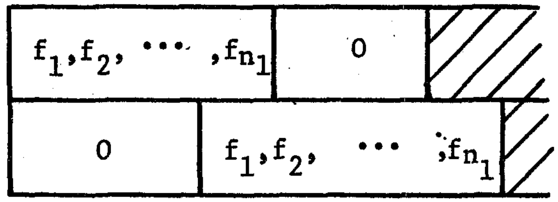
$$C_1(f_i) \leqq CF^*-B_F+\sum_{j=1}^{i-1} b_{f_j} < B_O+\sum_{j=1}^{i-1} b_{f_j}=S_2(f_i)$$

holds also in this case.

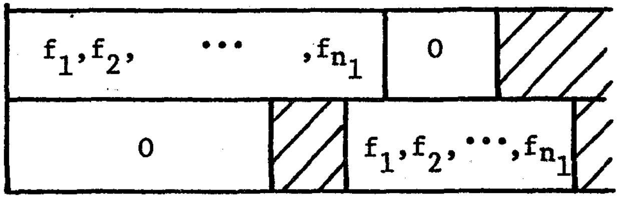
No idle time exists on M_1 and M_2, and this means $C_1=A_F+A_O$ and $C_2=B_O+B_F$. Thus, $C_{max}=\max(A_F+A_O, B_O+B_F)$.



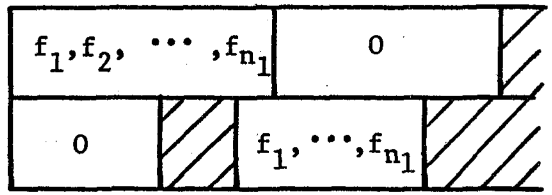
(a) $B_0 \geq CF^*$ and $A_F + A_0 \geq B_F + B_0$.



(b) $B_0 \geq CF^* - B_F$ and $A_F + A_0 < B_F + B_0$.



(c) $B_0 < CF^* - B_F$ and $CF^* \geq A_F + A_0$.



(d) $B_0 < CF^* - B_F$ and $CF^* < A_F + A_0$.

Fig. 2.9. The typical schedules for Case 1.

Consequently, if $A_F \geqq B_0$, by Lemma 2.6 we can show that the schedule constructed by algorithm I is an optimal schedule. $\square$

Note that for the above case, we can obtain the optimal schedule regardless of $B_F \geqq A_0$ or $B_F < A_0$.

Case 2: $B_F \geqq A_0$ and $A_F < B_0$

We develop the algorithm for $B_F \geqq A_0$ and $A_F < B_0$.

Algorithm II

(1) On M_1, process the jobs in F successively in an arbitrary order from time 0. Then, process the jobs in 0 successively in an arbitrary order from time B_0.

(2) On M_2, process the jobs in 0 first and next the jobs in F successively in an arbitrary order from time 0.

In this case, the typical schedule is the one illustrated in Fig. 2.10.

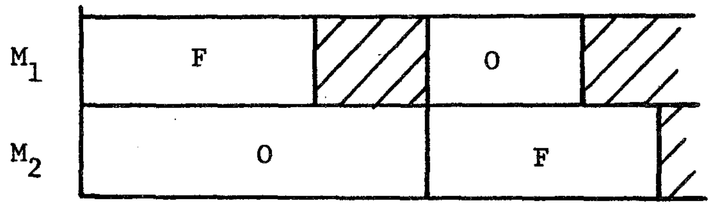


Fig. 2.10. The typical schedule for Case 2.

Lemma 2.8. If $B_F \geqq A_0$ and $A_F < B_0$, then algorithm II constructs an optimal schedule.

Proof. For any $i \in F$, it is clear that $C_1(i) \leqq A_F$ and $S_2(i) \geqq B_0$. Since $A_F < B_0$, we have $C_1(i) < S_2(i)$. For any $i \in 0$, we obtain $C_2(i) \leqq B_0 \leqq S_1(i)$. Moreover, it is clear that $C_1 = B_0 + A_0$ and $C_2 = B_0 + B_F$. Since $A_0 \leqq B_F$, we get $C_{max} = B_0 + B_F$. Thus, from Lemma 2.6, algorithm II constructs the optimal schedule. $\square$

Case 3: $B_F < A_0$ and $A_F < B_0$

We divide this case into some subcases and develop the algorithm for each subcase.

(I) $A_0 - a_r \leqq B_0 - b_l$

Let $0' = 0 - \{r\}$, $A_{0'} = \sum_{i \in 0'} a_i$, $T_1 = B_0 - A_{0'}$, and $T_2 = a_r$.

(I-1) Subcase 1: $T_1 \leqq A_F$

Algorithm III

(1) On M_1, process the jobs in F successively in an arbitrary order from time 0. Then, process the jobs in 0 successively in the order $s_1, s_2, \cdots, s_{n_2-1}$ and s_{n_2} from time A_F.

(2) On M_2, process tha jobs in 0 successively in the order $s_1, s_2, \cdots, s_{n_2-1}$ and s_{n_2} from time 0. Then, process the jobs in F successively in an arbitrary order from time B_0.

The schedules characterizing this case are illustrated in Fig. 2.11.

Lemma 2.9. If $A_0 - a_r \leqq B_0 - b_l$ and $T_1 \leqq A_F$, then the schedule generated by algorithm III is an optimal schedule.

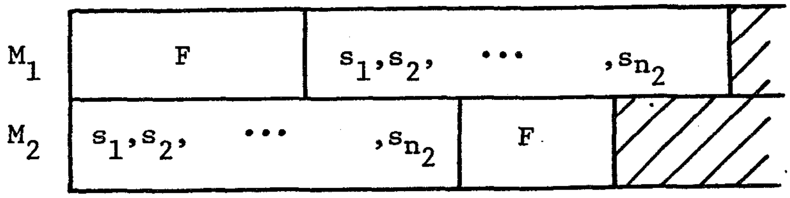
Proof. Since for any $i \in F$, we get

$$C_1(i) \leqq A_F$$

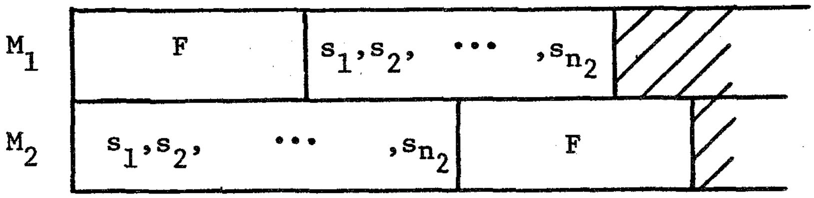
and

$$S_2(i) \geqq B_0,$$

from the assumption $A_F < B_0$ we can prove $C_1(i) < S_2(i)$. Further,



(a) $A_F + A_0 \geq B_F + B_0$.



(b) $A_F + A_0 < B_F + B_0$.

Fig. 2.11. The typical schedules for Subcase 1.

for any $s_i \epsilon O$, it is easy to show $C_2(s_i) = \sum_{j=1}^{i} b_{s_j}$ and $S_1(s_i) = A_F + \sum_{j=1}^{i-1} a_{s_j}$. By arranging the equation of $S_1(s_i)$, we have

$$
\begin{aligned}
S_1(s_i) &= A_F + \sum_{j=1}^{i-1} a_{s_j} = A_F + A_{0'} - \sum_{j=i}^{n_2-1} a_{s_j} \\
&= A_F - (B_0 - A_{0'}) + B_0 - \sum_{j=i}^{n_2-1} a_{s_j} \\
&= A_F - T_1 + B_0 - \sum_{j=i}^{n_2-1} a_{s_j}.
\end{aligned}
$$

Thus we obtain

$$S_1(s_i)-C_2(s_i)=A_F-T_1-\sum_{j=i}^{n_2-1} a_{s_j}+B_0-\sum_{j=i}^{i} b_{s_j}$$

$$=A_F-T_1+\sum_{j=i+1}^{n_2} b_{s_j}-\sum_{j=i}^{n_2-1} a_{s_j}.$$

Consequently, from $A_F \geqq T_1$ and Lemma 2.4 we can show that $S_1(s_i) \geqq C_2(s_i)$. On the other hand, there exists no idle time between consecutive jobs. Thus, it is clear that $C_1=A_F+A_0$ and $C_2=B_F+B_0$. Therefore, since the maximum completion time is $C_{max}=(A_F+A_0, B_F+B_0)$, Lemma 2.9 follows from Lemma 2.6. □

(I-2) Subcase 2: $T_1 > A_F$ and $T_2 \leqq B_F$

Note that in Subcase 1, we can obtain an optimal schedule regardless of $T_2 \leqq B_F$ or $T_2 > B_F$.

Algorithm IV

(1) On M_1, process the jobs in F continuously in an arbitrary order from time 0. Then, process the jobs in 0 successively in the order $s_1, s_2, \cdots, s_{n_2-1}$ and s_{n_2} from time T_1.

(2) On M_2, process the jobs in 0 without interruption in the order $s_1, s_2, \cdots, s_{n_2-1}$ and s_{n_2}. Next, process the jobs in F continuously in an arbitrary order from time B_0.

The only typical schedule for this case is shown in Fig. 2.12.

Lemma 2.10. If $A_0-a_r \leqq B_0-b_l$, $T_1 > A_F$ and $T_2 \leqq B_F$, then the schedule constructed by algorithm IV is an optimal schedule.

Proof. For any $s_i \in 0$, it is easy to show

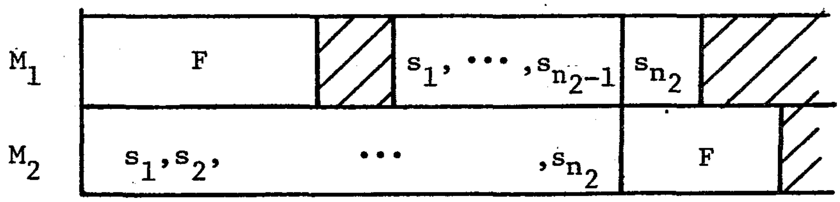


Fig. 2.12. The typical schedule for Subcase 2.

$$C_2(s_i)=\sum_{j=1}^{i} b_{s_j}$$

and

$$S_1(s_i)=T_1+\sum_{j=1}^{i-1} a_{s_j}=B_0-A_{0'}+\sum_{j=1}^{i-1} a_{s_j}$$

$$=B_0-\sum_{j=i}^{n_2-1} a_{s_j}.$$

From Lemma2.4 we have $S_1(s_i)\geqq C_2(s_i)$. For any $i\in F$, we can show that $C_1(i)\leqq T_1\leqq B_0\leqq S_2(i)$. On the other hand, we have $C_1=T_1+A_0=B_0+a_r=B_0+T_2$ and $C_2=B_0+B_F$. Since $B_F\geqq T_2$, we can show that $C_{max}=B_0+B_F$. Therefore, Lemma 2.10 follows from Lemma 2.6. □

(I-3) Subcase 3: $T_1 > A_F$ and $T_2 > B_F$

<u>Algorithm V</u>

(1) On M_1, process the jobs in $0'$ successively in the order $s_{n_2-1}, s_{n_2-2}, \cdots, s_2$ and s_1 from time 0. Then, process the jobs in F successively in an arbitrary order from time $A_{0'}$. Finally, process job r from time $\max(A_{0'}+A_F,\ b_r)$.

(2) On M_2, process the jobs in $0'$ successively in the order of $s_{n_2}, s_{n_2-1}, \cdots, s_2$ and s_1 from time 0. Then, process

the jobs in F without interruption in an arbitrary order from time B_0.

The typical schedules in this case are illustrated in Fig. 2. 13.

Lemma 2.11. If $A_0-a_r \leqq B_0-b_l$, $T_1 > A_F$ and $T_2 > B_F$, then algorithm V constructs an optimal schedule.

Proof. (i) When $A_{0'}+A_F \leqq b_r$, it is easy to show $C_1(i) \leqq b_r \leqq S_2(i)$ for any $i \in O' \cup F$. When $i=r$, we get $C_2(r)=b_r$ and $S_1(r)=b_r$. Moreover, we have $C_1=a_r+b_r$ and $C_2=B_0+B_F$. Thus, $C_{max}=\max(a_r+b_r, B_0+B_F)$.

(ii) When $A_{0'}+A_F>b_r$, we obtain $C_1(s_i)=\sum_{j=i}^{n_2-1} a_{s_j}$ and $S_2(s_i)=\sum_{j=i+1}^{n_2} b_{s_j}$ for any $s_i \in O'$. By Lemma 2.4, we can prove $C_1(s_i) \leqq S_2(s_i)$.

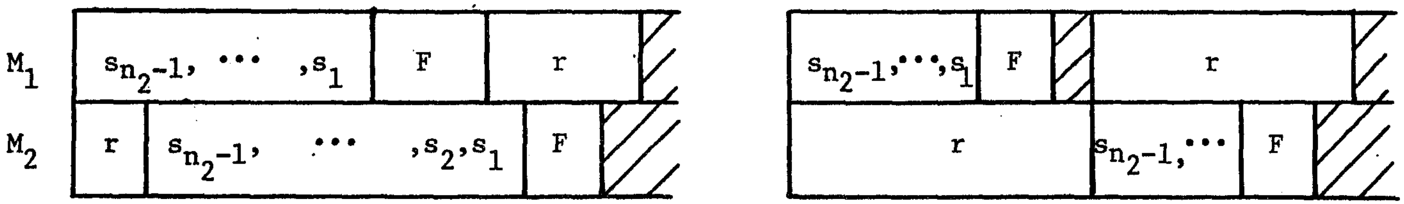
Further, for $s_i=r$, we have $C_2(r)=b_r$ and $S_1(r)=A_{0'}+A_F$. Thus the assumption $A_{0'}+A_F>b_r$ implies $C_2(r)<S_1(r)$. For any $i \in F$, we also have $C_1(i) \leqq A_{0'}+A_F$ and $S_2(i) \geqq B_0$. Since $A_{0'}+A_F<A_{0'}+T_1=B_0$, we have $C_1(i) \leqq S_2(i)$. Further, it is clear that $C_1=A_0+A_F$ and $C_2=B_0+B_F$. Consequently, $C_{max}=(A_0+A_F, B_0+B_F)$ for this case. Thus from Lemma 2.6, (i) and (ii) together the proof of this lemma is completed. □

(II) $A_0-a_r > B_0-b_l$

Hereafter, we change the definitions of O', $A_{0'}$, T_1 and T_2 as follows; $O'=O-\{l\}$, $A_{0'}=A_0-a$, $T_1=b_l$ and $T_2=A_0-B_{0'}$, where $B_{0'}=B_0-b_l$.

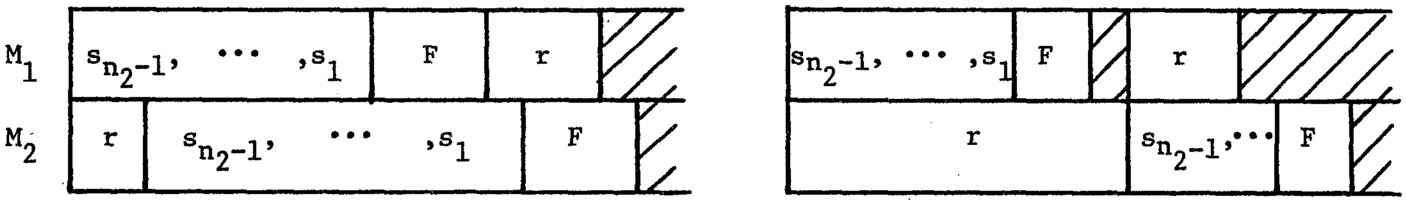
(II-1) Subcase 4: $T_1 \leqq A_F$

This subcase is the same as Subcase 1 except for the above



(a) $A_{0'}+A_F \geq b_r$ and $A_0+A_F \geq B_0+B_F$

(b) $b_r > A_{0'}+A_F$ and $a_r+b_r > B_0+B_F$



(a) $A_{0'}+A_F \geq b_r$ and $A_0+A_F < B_0+B_F$

(d) $b_r \geq A_{0'}+A_F$ and $a_r+b_r < B_0+B_F$

Fig. 2.13. The typical schedules for Subcase 3.

change of some definitions.

(II-2) Subcase 5: $T_1 > A_F$ and $T_2 \leq B_F$

The optimal schedule for this subcase can be obtained in the same way as for Subcase 2 except for the above change of some definitions.

(II-3) Subcase 6: $T_1 > A_F$ and $T_2 > B_F$

Algorithm VI

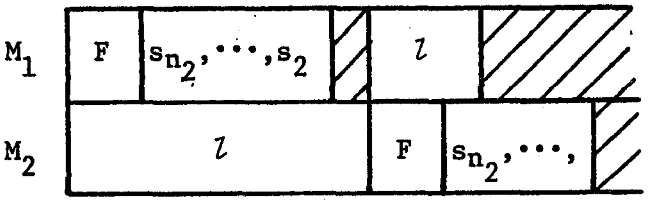
(1) On M_1, process the jobs in F successively in an arbitrary order from time 0. Then, process the jobs in O' continuously in the order $s_{n_2}, s_{n_2-1}, \cdots, s_3$ and s_2 from time A_F. Finally, process job $s_1(=l)$ from time max(b_l, $A_F+A_{O'}$).

(2) On M_2, process job l from time 0. Then, process the jobs in F without interruption in an arbitrary order from time b_l. Finally, process the jobs in O' successively in the order $s_{n_2}, \cdots, s_3$ and s_2 from time b_l+B_F if $b_l \geq A_F+A_{O'}$, or from time $\max(b_l+B_F, A_F+T_2)$ if $b_l < A_F+A_{O'}$.

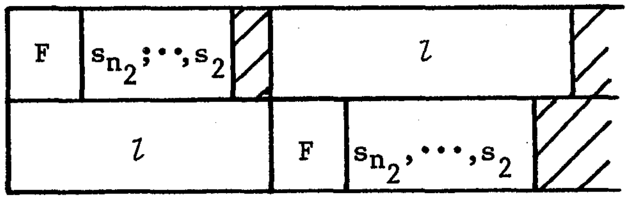
The typical schedules characterizing this case are illustrated in Fig. 2.14.

Lemma 2.12. If $A_O-a_r > B_O-b_l$, $T_1 > A_F$ and $T_2 > B_F$, then the schedule constructed by algorithm VI is an optimal schedule.

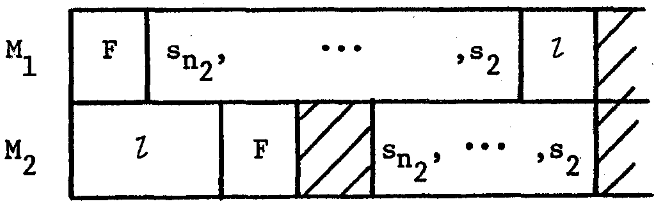
Proof. (i) When $b_l \geq A_F+A_{O'}$, we can easily show that for any $i \in F$, $C_1(i) \leq A_F \leq T_1 = b_l \leq S_2(i)$ holds. Further, it holds that for any $s_i \in O'$, $C_1(s_i) \leq A_F+A_{O'} \leq b_l$ and $S_2(s_i) \geq b_l+B_F$. Therefore we have $S_2(s_i) \geq C_1(s_i)$. Moreover, for $s_i=l$, $C_2(l)+b_l$ and $S_1(l)=b_l$ hold. Also, it is easy to show that $C_1=a_l+b_l$ and $C_2=B_O+B_F$. Accordingly,



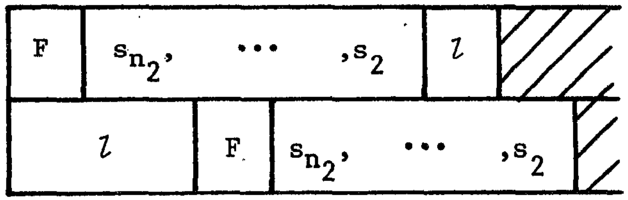
(a) $b_l \geq A_F + A_0$, and $a_l + b_l \leq B_0 + B_F$.



(b) $b_l \geq A_F + A_0$, and $a_l + b_l > B_0 + B_F$.



(c) $b_l < A_F + A_0$, and $A_F + A_0 \geq B_F + B_0$.



(d) $b_l < A_F + A_0$, and $A_F + A_0 < B_F + B_0$.

Fig. 2.14. The typical schedules for Subcase 6.

$C_{max}=\max(a_l+b_l,\ B_0+B_F)$.

(ii) When $b_l<A_F+A_{0'}$ and $b_l+B_F \geqq A_F+T_2$, we get $C_1(i)\leqq A_F \leqq T_1=b_l \leqq S_2(i)$ for any $i\in F$. Further, for any $s_i \in O'$, we have

$$C_1(s_i)=A_F+\sum_{j=i}^{n_2} a_{s_j}=A_F+A_0-\sum_{j=1}^{i-1} a_{s_j}=A_F+T_2+B_{0'}-\sum_{j=1}^{i-1} a_{s_j},$$

and

$$S_2(s_i)=B_F+b_l+\sum_{j=i+1}^{n_2} b_{s_j}=B_F+B_0-\sum_{j=1}^{i} b_{s_j}+b_l$$

$$\geqq B_F+b_l+B_{0'}-\sum_{j=1}^{i} b_{s_j}+b_l.$$

Thus

$$S_2(s_i)-C_1(s_i)=(B_F+b_l)-(A_F+T_2)+b_l+\sum_{j=1}^{i-1} a_{s_j}-\sum_{j=1}^{i} b_{s_j}$$

holds. Therefore from Lemma 2.5 and the assumption $A_F+T_2 \leqq B_F+b_l$, we can prove that $S_2(s_i)\geqq C_1(s_i)$. For $s_i=l$, since $C_2(l)=b_l$ and $S_1(l)=A_F+A_{0'}$, clearly $C_2(l)\leqq S_1(l)$ holds. On the other hand, it is obtained that $C_1=A_F+A_{0'}+a_l=A_F+A_0$ and $C_2=b_l+B_F+B_{0'}=B_F+B_0$. Since $b_l+B_F \geqq A_F+T_2=A_F+A_0-B_{0'}$, we have $B_0+B_F \geqq A_F+A_0$. Accordingly, $C_{max}= B_0+B_F$.

(iii) When $b_l<A_F+A_{0'}$ and $b_l+B_F<A_F+T_2$, for any $i\in F$ we can easily show that $C_1(i)\leqq A_F \leqq b_l \leqq S_2(i)$. Also for any $s_i \in O'$, we get

$$C_1(s_i)=A_F+\sum_{j=i}^{n_2} a_{s_j}=A_F+A_0-\sum_{j=1}^{i-1} a_{s_j},$$

and

$$S_2(s_i)=A_F+T_2+\sum_{j=i+1}^{n_2} b_{s_j}=A_F+A_0-B_{0'}+\sum_{j=i+1}^{n_2} b_{s_j}$$

$$=A_F+A_0-\sum_{j=1}^{i} b_{s_j}+b_l.$$

Thus the equality

$$S_2(s_i)-C_1(s_i)=b_l+\sum_{j=1}^{i-1} a_{s_j}-\sum_{j=i}^{i} b_{s_j}$$

holds. Therefore from Lemma 2.5, we can prove $S_2(s_i)\geq C_1(s_i)$. Further, for $s_i=l$, we have $C_2(l)=b_l<A_F+A_{0'}=S_1(l)$. Also, since $C_1=A_F+A_0$ and $C_2=T_2+A_F+B_{0'}=A_0+A_F$, we have $C_{max}=A_F+A_0$. Consequently (i), (ii) and (iii) prove this lemma. □

We present the complete scheduling algorithm before our main theorem.

Complete Algorithm

Step 0. Set $A_F=\sum_{i\in F} a_i$, $B_F=\sum_{i\in F} b_i$, $A_0=\sum_{i\in 0} a_i$ and $B_0=\sum_{i\in 0} b_i$.

Step 1. If $A_F\geq B_0$, then go to Step 2. Otherwise go to Step 3.

Step 2. (1) On M_1, process the jobs in F successively according to Johnson's rule from time 0. Then process the jobs in 0 successively in an arbitrary order from time A_F after processing all flow shop type jobs.

(2) On M_2, process the jobs in 0 successively in an arbitrary order from time 0. Then, process the jobs in F according to Johnson's rule from time $\max(B_0,\ CF^*-B_F)$. (Algorithm I)

Step 3. If $B_F\geq A_0$, then go to Step 4. Otherwise go to Step 5.

Step 4. (1) On M_1, process the jobs in F successively in an arbitrary order from time 0. Then, process the jobs in 0 successively in an arbitrary order from time B_0.

(2) On M_2, process the jobs in 0 first and then the

jobs in F successively in an arbitrary order from time 0. (Algorithm II)

Step 5. Find any two distinct jobs r and l in O such that $b_r \geq \max_{j \in O_2} (a_j)$ and $a_l \geq \max_{j \in O_1} (b_j)$. If $A_O - a_r \leq B_O - b_l$, then set $O' = O - \{r\}$, $A_{O'} = A_O - a_r$, $T_1 = B_O - A_{O'}$ and $T_2 = a_r$. Otherwise, set $O' = O - \{l\}$, $A_{O'} = A_O - a_l$, $B_{O'} = B_O - b_l$, $T_1 = b_l$ and $T_2 = A_O - B_{O'}$.

Step 6. If $T_1 \leq A_F$ then go to Step 7. Otherwise, go to Step 8.

Step 7. (1) On M_1, process the jobs in F successively in an arbitrary order from time 0. Then, process the jobs in O successively the order s_1, $s_2, \cdots, s_{n_2-1}$ and s_{n_2} from time A_F.

(2) On M_2, process the jobs in O successively in the order $s_1, s_2, \cdots, s_{n_2-1}$ and s_{n_2} from time 0. Then, process the jobs in F successively in an arbitrary order from time B_O.

(Algorithm III)

Step 8. If $T_2 \leq B_F$, then go to Step 9. Otherwise go to Step 10.

Step 9. (1) On M_1, process the jobs in F continuously in an arbitrary order from time 0. Then, process the jobs in O successively in the order $s_1, s_2, \cdots, s_{n_2-1}$ and s_{n_2} from time T_1.

(2) On M_2, process the jobs in O without interruption in the order $s_1, s_2, \cdots, s_{n_2-1}$ and s_{n_2} from time 0. Next, process the jobs in F continuously in an arbitrary order from time B_O. (Algorithm IV)

Step 10. If $A_0-a_r \leq B_0-b_l$, then go to Step 11. Otherwise, go to Step 12.

Step 11. (1) On M_1, process the jobs in O' successively in the order $s_{n_2-1}, s_{n_2-2}, \cdots, s_2$ and s_1 from time 0. Then, process the jobs in F successively in an arbitrary order from time $A_{0'}$. Finally, process job r from time $\max(A_{0'}+A_F, b_r)$.

(2) On M_2, process the jobs in O successively in the order of $s_{n_2}, s_{n_2-1}, \cdots, s_2$ and s_1 from time 0. Then, process the jobs in F without interruption in an arbitrary order from time B_0. (Algorithm V)

Step 12. (1) On M_1, process the jobs in F successively in an arbitrary order from time 0. Then, process the jobs in O' continuously in the order of $s_{n_2}, s_{n_2-1}, \cdots, s_3$ and s_2 from time A_F. Finally, process job $s_1(=l)$ from time $\max(b_l, A_F+A_{0'})$.

(2) On M_2, process job l from time 0. Then, process the jobs in F without interruption in an arbitrary order from time b_l. Finally, process the jobs in O' successively in the order $s_{n_2}, \cdots, s_3$ and s_2 from time b_l+B_F if $b_l \geq A_F+A_{0'}$, or from time $\max(b_l+B_F, A_F+T_2)$ if $b_l < A_F+A_{0'}$. (Algorithm VI)

Using Lemmas 2.7-2.12, the next main theorem is deduced.

Theorem 2.3. If in a two machine scheduling problem there are flow shop type jobs and open shop type jobs, then above Complete algorithm (or algorithms I-VI) gives an optimal schedule minimizing the maximum completion time. $\square$

CHAPTRE 3

SCHEDULING PROBLEMS ON PARALLEL TYPE MACHINES

3.1 Introduction

In this chapter, we consider scheduling problems for a set of jobs $J=\{J_1,J_2,\cdots,J_n\}$ to be processed on parallel type machines. The following three problems are dealt with.

(i) $n|m|I|L_{max}$ nonpreemptive scheduling problem: Each job J_i consists of single operation which can be processed on any machine and has an equal processing time on each machine, i.e., $p_{ij}=p_i$, and so machines are identical parallel type. Further, each job J_i has a same due date on each machine M_j, i.e., $d_{ij}=d_i$. The processing of J_i should ideally be completed within the due date. The schedule must be nonpreemptive and the objective is to minimize maximum lateness.

(ii) $n|2|I|L_{max}$ preemptive scheduling problem with generalized due dates: Each job J_i is to be processed on two identical parallel machines. On machine $M_1(M_2)$ J_i must be completed by the due date $d_{1i}(d_{2i})$. The objective is to minimize maximum lateness.

(iii) $n|m|QI|C_{max}$ scheduling problem: Each job J_i can be processed only on a given subset of machine set M, $Q_i=\{M_j \mid j\in I_i\}$, where $I_i=\{h_i(1),\cdots,h_i(k_i)\}\subseteq I=\{1,2,\cdots,m\}$. The processing time of J_i on machine M_j is $p_{ij}=p_i$ if $M_j\in Q_i$ and $p_{ij}=\infty$ if $M_j\notin Q_i$. The objective is to obtain a preemptive or nonpreemptive schedule minimizing the maximum completion time.

For the nonpreemptive scheduling on parallel type machines, above three types of problems are NP-complete except for the cases that each job has unit processing time or the objective is to minimize total completion time $\sum_{i=1}^{n}C_i$. For the preemptive case, the following facts are already known.

machine type	objective	complexity	reference
single	L_{max}	$O(n\log n)$	[11]
identical	C_{max}	$O(n)$	[23]
identical	L_{max}	$O(n^2)$	[10]
uniform	C_{max}	$O(n)$	[6]
uniform	L_{max}	$O(n^2)$	[25]

In Section 3.2, we propose two approximation algorithms for $n|m|I|L_{max}$ nonpreemptive scheduling problem, which is NP-complete, and show their worst case bounds.

In Section 3.3, we present a polynomial time algorithm to construct a schedule minimizing maximum lateness for $n|2|I|L_{max}$ preemptive scheduling problem with generalized due dates.

Finally, in Section 3.4, we show a solvable case of $n|m|QI|C_{max}$ nonpreemptive scheduling problem and present a polynomial time algorithm for $n|m|QI|C_{max}$ preemptive scheduling problem. In such a solvable case, we assume that each job J_i has unit processing time.

3.2 Approximation Algorithms for $n|m|I|L_{max}$ Nonpreemptive Scheduling Problem and Their Worst Case Bounds

In this section, we discuss an $n|m|I|L_{max}$ nonpreemptive scheduling problem. In this problem, each job $J_i (\in J)$ has an equal processing time p_i and an equal due date d_i on each machine $M_j (\in M)$. The objective is to construct a schedule minimizing maximum lateness.

Unfortunately, this problem belongs to a class of NP-complete problems. Therefore, we propose two approximation algorithms and obtain their worst case bounds. Concerning the worst case bound, we use the form of modified relative deviation defined in Section 2.2.

Our first algorithm *EDD(Earliest Due Date)* is a list scheduling and the second algorithm *LPT(Longest Processing Time)* is its refinement.

List Scheduling: A *list scheduling* produces a schedule of jobs based on a list as follows. When one of the machines becomes available, first unprocessed job on the list is assigned to this machine.

In the list scheduling, the resulting schedule is influenced by the ordering of jobs on the list. Therefore we have to specify an ordering in advance. R. L. Graham [7] [8] obtained the following result with respect to a nonpreemptive schedule minimizing maximum completion time on m identical parallel machines ($n|m|I|C_{max}$ nonpreemptive scheduling problem).

Lemma (Graham [7] [8]). For $n|m|I|C_{max}$ nonpreemptive scheduling problem, let C'_{max} be the maximum completion time of any list scheduling and C^*_{max} that of optimal scheduling. Then, the inequality

$$\frac{C'_{max}}{C^*_{max}} \leqq 2 - \frac{1}{m}$$

holds [7]. Further, for the list on which the jobs are ordered in nonincreasing order of processing times, we have

$$\frac{C'_{max}}{C^*_{max}} \leqq \frac{4}{3} - \frac{1}{3m}$$

([8]).

For the job set J, the maximum lateness of the schedule constructed by some algorithm π (approximation or exact) is defined by

$$L(J;\pi) = \max_{1 \leqq i \leqq n} \{C_i(\pi) - d_i\},$$

where $C_i(\pi)$ is the corresponding completion time of J_i in that schedule. Especially, hereafter, the notations L(J;EDD), L(J;LPT) and $L(J;\pi^*)$ are used to denote the maximum latenesses for EDD, LPT and a certain optimum algorithm π^*, respectively.

3.2.1 Approximation algorithm EDD and its worst case bound

We present an approximation algorithm EDD and give its worst case bound (or modified relative deviation). Here, without any loss of generality, we can assume $d_1 \leqq d_2 \leqq \cdots \leqq d_n$.

Algorithm EDD: Assign the jobs to machines in the order, $J_1, J_2, \cdots, J_n$.

Theorem 3.1. For any job set J, the inequality

$$\frac{L(J;EDD)-L(J;\pi^*)}{L(J;\pi^*)+d_{max}} \leqq 1-\frac{1}{m}$$

holds, where $d_{max}=d_n$ from the above assumption. Moreover, this bound is the best possible.

Proof. We assume that a job set J is the smallest one for which the theorem may be violated. And, it is enough to consider only the case that job J_n determines the maximum lateness of the schedule constructed by algorithm EDD, i.e.,

(3.1) $\quad L(J;EDD)=C_n(EDD)-d_n.$

Since algorithm EDD is a list scheduling in which the jobs on the list are ordered in the nondecreasing order of due dates, Graham's Lemma shows that

(3.2) $$\frac{C_n(EDD)}{C(\bar{\pi}^*)} \leqq 2-\frac{1}{m},$$

where $\bar{\pi}^*$ is a certain exact algorihm minimizing maximum completion time and $C(\bar{\Pi}^*)$ is its maximum completion time. From (3.2), the inequality

(3.2)' $\quad C_n(EDD)\leqq(2-\frac{1}{m})C(\bar{\Pi}^*)$

holds. Substituting (3.2)' into (3.1), we obtain

(3.3) $\quad L(J;EDD)\leqq(2-\frac{1}{m})C(\bar{\Pi}^*)-d_n.$

On the other hand, we have

(3.4) $\quad L(J;\pi^*)\geqq C(\bar{\pi}^*)-d_n.$

Hence (3.3) and (3.4) imply that

$$(3.5) \qquad L(J;EDD)-L(J;\pi^*) \leqq (2-\frac{1}{m})C(\bar{\pi}^*)-d_n-(C(\bar{\pi}^*)-d_n)$$

$$=(1-\frac{1}{m})C(\bar{\pi}^*).$$

Since $d_{max}=d_n$, (3.4) and (3.5) together show that

$$\frac{L(J;EDD)-L(J;\pi^*)}{L(J;\pi^*)+d_{max}} \leqq \frac{(1-\frac{1}{m})C(\bar{\pi}^*)}{C(\bar{\pi}^*)} = 1-\frac{1}{m}.$$

This contradicts our assumption. Thus, we have the desired worst case bound.

To see that this bound is the best possible, we can consider three examples depending on m (mod 4).

Example 1. Let $n=2m+1$ and $m=2r$. The processing times are given by

$$p_{2i-1}=p_{2i}=\begin{cases} r+(i-1) & \text{for } 1\leqq i\leqq r \\ 4r-(i+1) & \text{for } r+1\leqq i\leqq 2r \end{cases}$$

and $p_{4r+1}=4r$, and the due dates are $d_i=d$(=const.) for $1\leqq i\leqq 2m+1$. Since all the due dates are equal, we may assume that the i-th job assigned by algorithm EDD, $1\leqq i\leqq 2m+1$, is job J_i. Then, we obtain the schedule shown in Fig. 3.1(a). Because the optimal schedule by some exact algorithm π^* becomes as shown in Fig.3.1(b), and $L(J;\pi^*)$ is $2m-d$, we have

$$\frac{L(J;EDD)-L(J;\pi^*)}{L(J;\pi^*)+d_{max}} = 1-\frac{1}{2r} = 1-\frac{1}{m}.$$

Example 2. Let $n=2m+1$ and $m=4r+1$ and let the processing times be given by

$$p_i = \begin{cases} 2r+2\lceil \frac{i}{4} \rceil -1 & \text{for } 1 \leq i \leq 4r, \\ 4r & \text{for } i=4r+1, \\ 8r-2\lceil \frac{i-1}{4} \rceil +1 & \text{for } 4r+2 \leq i \leq 8r+1, \\ 4r & \text{for } i=8r+2, \\ 8r+2 & \text{for } i=8r+3, \end{cases}$$

where $\lceil x \rceil$ is minimum integer not less than x, and the due dates be $d_i=d$ for $1 \leq i \leq 2m+1$.

The approximate and the optimal schedules for this case are illustrated in Fig. 3.2(a) and (b), respectively. Similar to Example 1, we obtain $L(J;EDD)=16r+2-d$ and $L(J;\pi^*)=8r+2-d$. Thus, we have

$$\frac{L(J;EDD)-L(J;\pi^*)}{L(J;\pi^*)+d_{max}} = \frac{8r}{8r+2} = 1 - \frac{1}{m}.$$

Example 3. When $n=2m+1$ and $m=4r+3$, let the processing times and the due dates are given by

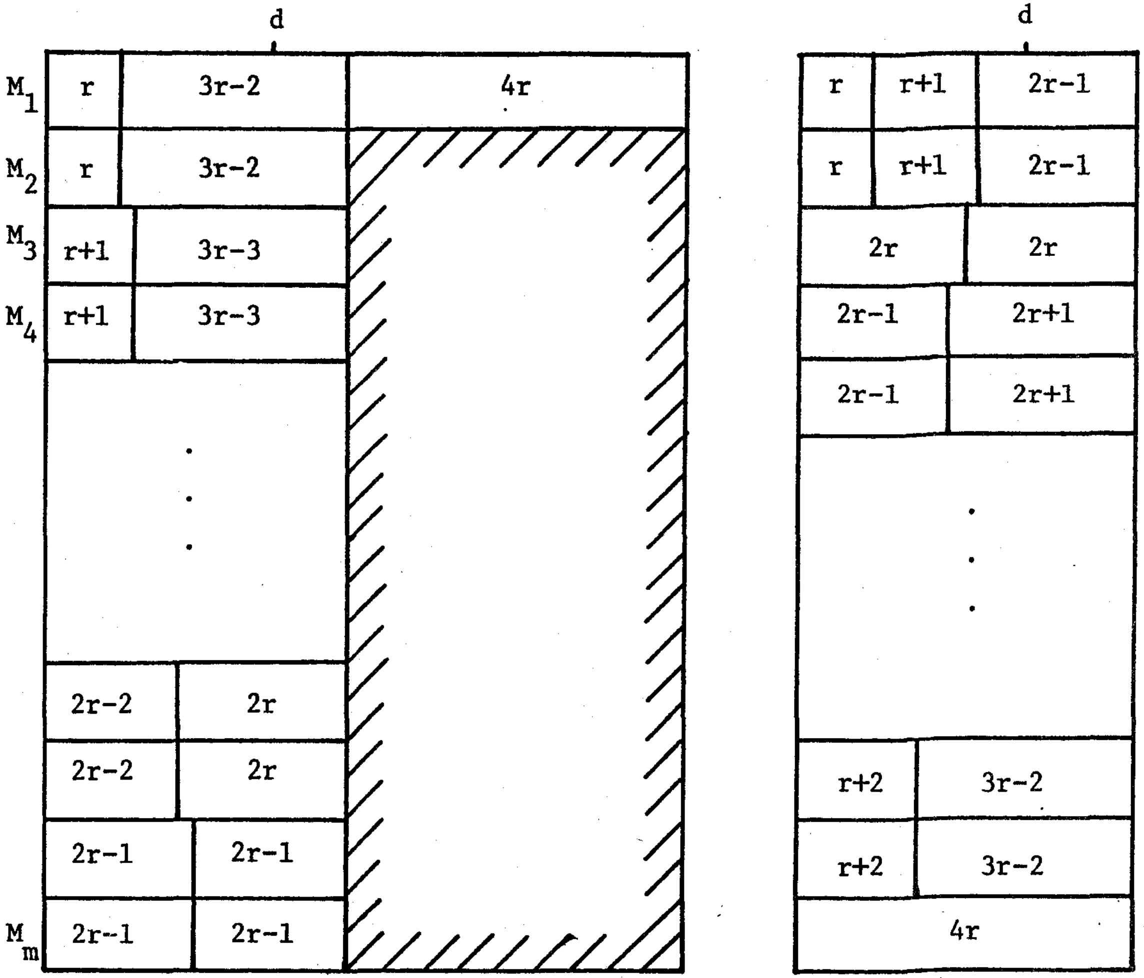
$$p_i = \begin{cases} r+\lceil \frac{i}{4} \rceil & \text{for } 1 \leq i \leq 4r, \\ 2r+1 & \text{for } i=4r+1, 4r+2, 4r+3, 8r+4, 8r+5, 8r+6, \\ 4r+2-\lfloor \frac{i}{4} \rfloor & \text{for } 4r+4 \leq i \leq 8r+3, \\ 4r+3 & \text{for } i=8r+7, \end{cases}$$

where $\lfloor x \rfloor$ is maximum integer not greater than x, and $d_i=d$ for $1 \leq i \leq 2m+1$.

The approximate and the optimal schedules are illustrated in Fig. 3.3(a) and (b), respectively. Again, similar to Examples 1 and 2, we have $L(J;EDD)=8r+5-d$ and $L(J;\pi^*)=4r+3-d$. Hence, we get

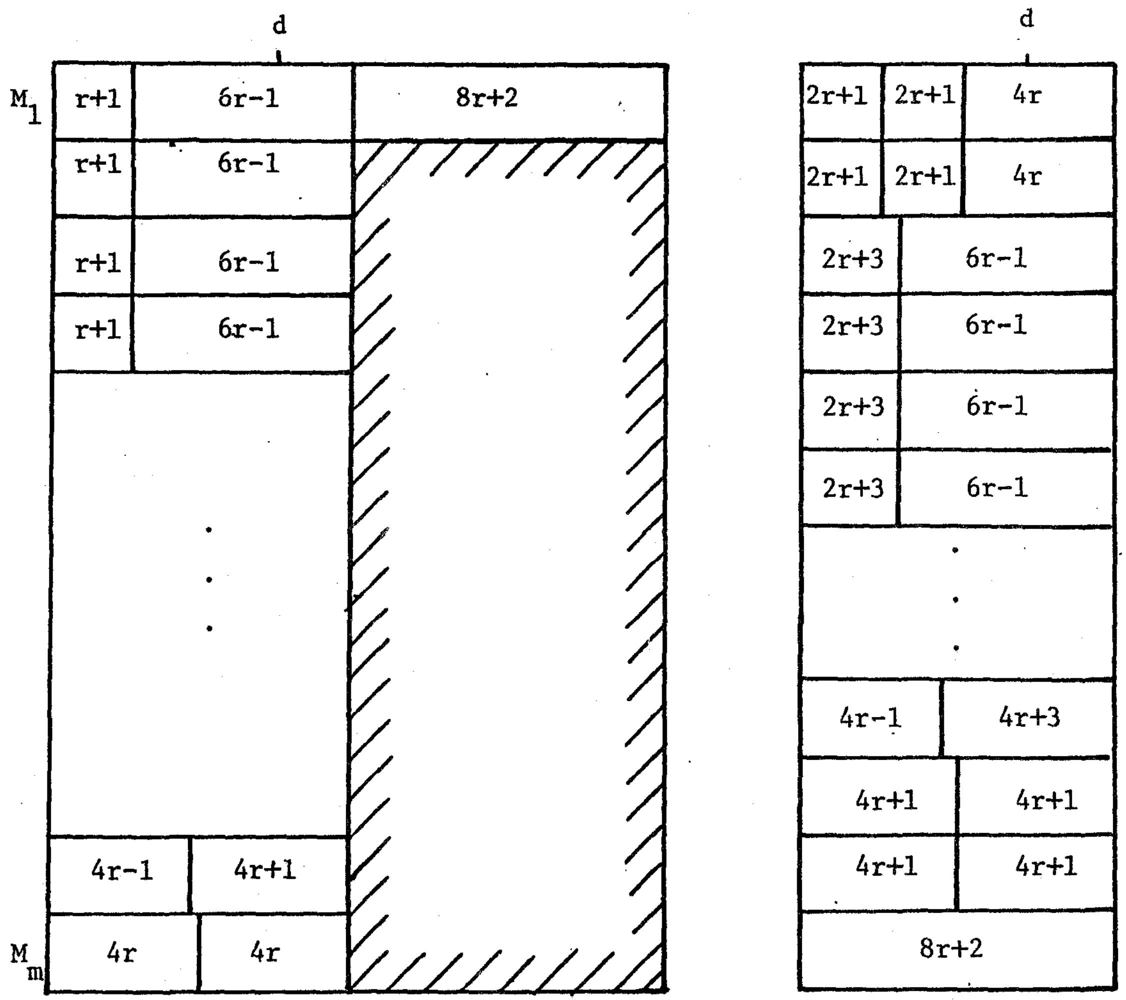
$$\frac{L(J;EDD)-L(J;\pi^*)}{L(J;\pi^*)+d_{max}} = \frac{4r+2}{4r+3} = 1 - \frac{1}{m}.$$

This completes the proof of Theorem 3.1. $\square$



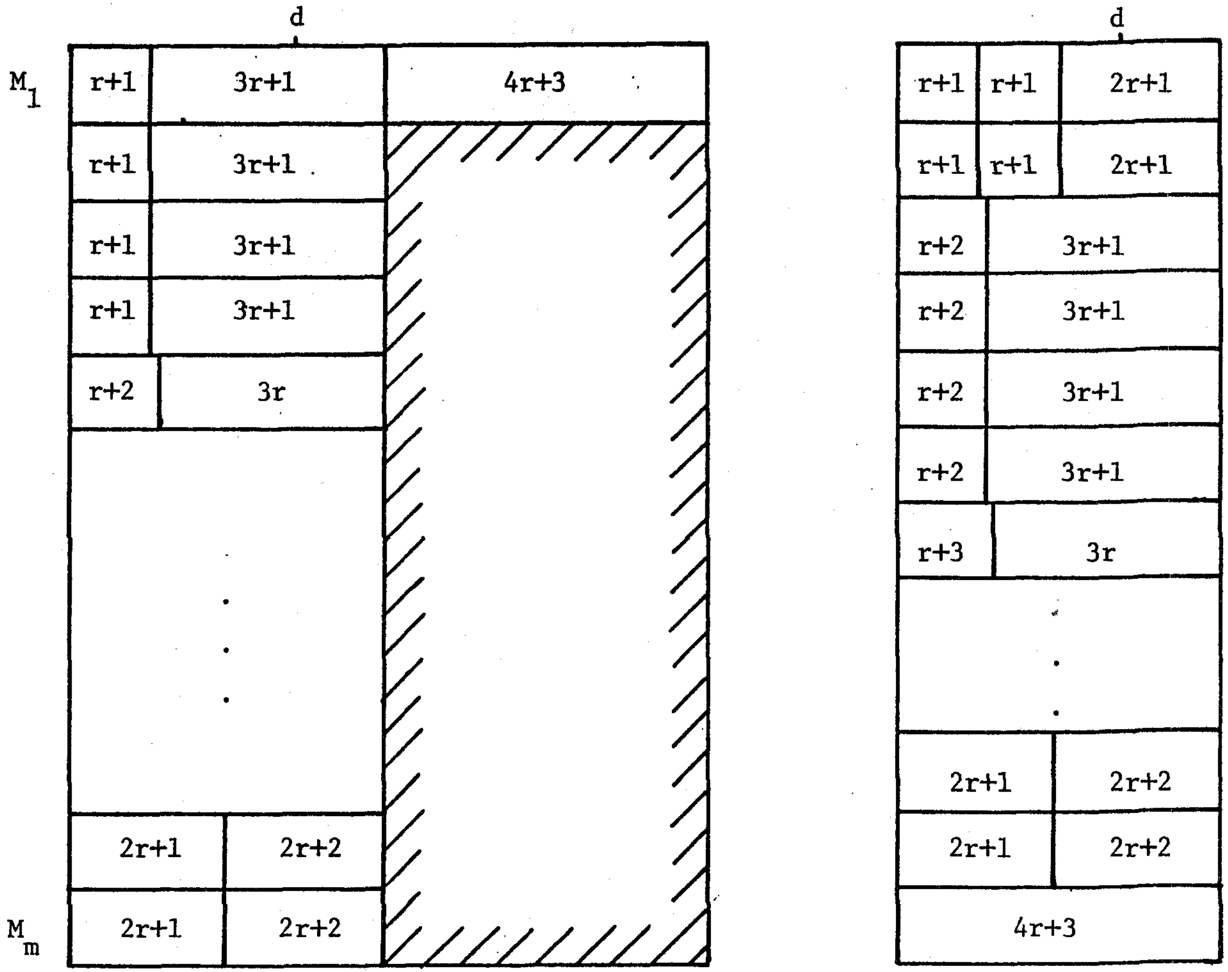
(a) Approximation algorithm (b) Optimal schedule

Fig. 3.1. An example giving the tight bound of Theorem 3.1 in case m=2r.



(a) Approximation schedule. (b) Optimal schedule.

Fig. 3.2. An example giving the tight bound of Theorem 3.1 in case m=4r+1.



(a) Approximation schedule. (b) Optimal schedule

Fig. 3.3. An example giving the tight bound of Theorem 3.1 in case m=4r+3.

3.2.2 Approximation algorithm LPT and its worst case bound

The worst case examples in the last subsection show that when the number of distinct due dates is small, the algorithm EDD is not so effective. In such a case, the maximum lateness may be greatly influenced by the maximum completion time rather than the due dates. Now, we propose another approximation algorithm LPT which is more effective in such a situation, and give the worst case bound. Algorithm LPT is a hybrid algorithm which consists of a mixture of LPT and EDD rules.

Algorithm LPT:

Step 1. Assign the jobs to each machine according to the list such that the jobs are ordered in the non-increasing order of processing times. (LPT rule)

Step 2. On each machine, reorder the assigned jobs according to the nondecreasing order of due dates. (EDD rule)

Next Theorem 3.2 gives a worst case bound of algorithm LPT. But probably algorithm LPT has a better worst case bound than that of Theorem 3.2 in some cases.

Theorem 3.2. Let $L(J;LPT)$ and $L(J;\pi^*)$ be the maximum latenesses of the schedules constructed by algorithm LPT and some exact algorithm π^* for job set J, respectively. Then,

$$\frac{L(J;LPT)-L(J;\pi^*)}{L(J;\pi^*)+d_{max}} \leq \min \left\{ \frac{4}{3} - \frac{1}{3m} - \frac{mp_{min}}{P},\ \frac{1}{3} - \frac{1}{3m} + \frac{m(d_n-d_1)}{P} \right\}$$

holds, where $p_{min} = \min_{1 \leq i \leq n} p_i$ and $P = \sum_{i=1}^{n} p_i$.

Proof. Let $\bar{\pi}^*$ be any exact algorithm minimizing maximum completion time for job set J and $C(\bar{\pi}^*)$ the maximum completion time of $\bar{\pi}^*$.

It is clear that the inequality

$$(3.6) \qquad L(J;\pi^*) \geqq C(\bar{\pi}^*) - d_n$$

holds. Also, we have

$$(3.7) \qquad L(J;LPT) \leqq C(LPT) - d_1,$$

where C(LPT) is the maximum completion time of the schedule constructed by algorithm LPT. From (3.6) and (3.7), we have

$$(3.8) \qquad \frac{L(J;LPT)-L(J;\pi^*)}{L(J;\pi^*)+d_{max}} \leqq \frac{C(LPT)-C(\bar{\pi}^*)}{C(\bar{\pi}^*)} + \frac{d_n - d_1}{C(\bar{\pi}^*)}.$$

Since

$$(3.9) \qquad \frac{C(LPT)}{C(\bar{\pi}^*)} \leqq \frac{4}{3} \ \frac{1}{3m}$$

by Graham's Lemma and $C(\bar{\pi}^*) \geqq P/m$, it holds that

$$(3.10) \qquad \frac{L(J;LPT)-L(J;\pi^*)}{L(J;\pi^*)+d_{max}} \leqq \frac{1}{3} - \frac{1}{3m} + \frac{m}{P}(d_n - d_1).$$

Further, let $L(J;LPT) = C_k - d_k$, where C_k is the completion time of job J_k in the schedule obtained by algorithm LPT. It is clear that

$$(3.11) \qquad L(J;\pi^*) \geqq p_{min} - d_k.$$

Since $C_k \leqq C(LPT)$, (3.9) and (3.11) imply that

$$L(J;LPT)-L(J;\pi^*) \leqq C_k - d_k - (p_{min} - d_k) \leqq C(LPT) - p_{min}$$

$$\leqq \left(\frac{4}{3} - \frac{1}{3m}\right) C(\bar{\pi}^*) - p_{min}.$$

From (3.6), we obtain

$$\frac{L(J;LPT)-L(J;\pi^*)}{L(J;\pi^*)+d_{max}} \leq \frac{(\frac{4}{3}-\frac{1}{3m})C(\bar{\pi}^*)-p_{min}}{C(\bar{\pi}^*)}$$

$$= \frac{4}{3} - \frac{1}{3m} - \frac{p_{min}}{C(\bar{\pi}^*)}$$

$$\leq \frac{4}{3} - \frac{1}{3m} - \frac{mp_{min}}{P}$$

Thus, we prove Theorem 3.2. $\square$

3.3 $n|2|I|L_{max}$ Preemptive Scheduling Problem with Generalized Due Dates

In this section, we consider an $n|2|I|L_{max}$ preemptive scheduling problem with generalized due dates. This problem is characterized as follows; (i) a set of jobs $J=\{J_1,J_2,\cdots,J_n\}$ is to be processed on two identical parallel type machines M_1 and M_2, (ii) processing time of each job J_i on M_1 or M_2 is p_i, (iii) each job J_i has a definite due date d_{ij} for machine M_j $(j=1,2)$, in other words, the processing of job J_i on M_j must be completed by the due date d_{ij}, (iv) preemptions for the jobs are admitted, and (v) our objective is to minimize the maximum lateness. (Note that $d_{i1}=d_{i2}$ is not necessary.)

In the last section, we proposed two approximation algorithms and obtained their worst case bounds for $n|m|I|L_{max}$ nonpreemptive scheduling problem. In that problem and other traditional scheduling problems with due dates, we assume that each job J_i must have a same due date on each machine, that is, $d_{ij}=d_{ij'}$, for $j\neq j'$ and $1\leq j,j'\leq m$. However, each job does not always have the same due date on each machine. For example, let A be a factory utilizing products by the completed job J_i. Then, the transportation times of goods by J_i from machines to A may differ and the actual due date is not a date to complete J_i on some machine but a date of delivery to A. Thus, the practical due date on each machine must differ for each machine. Therefore we generalize the idea of the due date in the sense that each job may have different due dates for each machine.

We consider the problem of obtaining a feasible schedule for given due dates.

Feasible Schedule: A *feasible schedule* for a job set is one such that all the jobs are completed by their due dates.

In the following subsections, we show how to reduce a problem of obtaining a feasible schedule to network flow problem and develop an efficient algorithm to get a feasible schedule.

3.3.1 Construction of associated network flow problem

We will construct a network flow problem corresponding to the feasible schedule for our present problem.

Let D_i, $1 \leq i \leq k$, denote the distinct values of due dates, where $D_1 < D_2 < \cdots < D_k$ and $k \leq 2n$. And, let $I_i = [D_{i-1}, D_i]$, $1 \leq i \leq k$, and $D_0 = 0$.

Now, we construct a network N with n+3k vertices, 2k of which are source vertices with labels m_{ji}, $j=1,2$, $i=1,2,\cdots,k$, corresponding to machines and time intervals I_i. The maximum possible amount of supply from each source m_{ji} is $s_i = D_i - D_{i-1}$. And, n vertices are sinks with labels $J_1,\cdots,J_n$ corresponding to the jobs. Each sink J_i has the demand p_i, $1 \leq i \leq n$. The remaining k vertices are intermediate ones labeled $v_1,\cdots,v_k$. Further, the network N contains three types of directed arcs. The first type is the arcs connecting source vertices to sink vertices. The second type is the ones from intermediate vertices to sinks. The last type is the ones from sources to intermediate vertices. If $d_{j1} \geq D_i$ and $d_{j2} < D_{i-1}$, arc (m_{1i}, J_j) connects vertex m_{1i} to J_j. If $d_{j1} < D_{i-1}$ and $d_{j2} \geq D_i$, arc (m_{2i}, J_j) connects vertex m_{2i} to J_j. And if $d_{j1} \geq D_i$ and $d_{j2} \geq D_i$, arc (v_i, J_j) connects vertex v_i to J_j. Finally, for j=1,2 and $1 \leq i \leq k$, arc (m_{ji}, v_i) connects vertex m_{ji} to v_i. Moreover, the above all arcs have the same capacity $s_i = D_i - D_{i-1}$. Note that if job J_j, $1 \leq j \leq n$, can not be processed on either or both of two machines in some interval I_i, $1 \leq i \leq k$, then there exists no arc connecting vertices m_{1i}, m_{2i} and v_i to vertex J_j on the above reduced network.

Feasible Flow: A *feasible flow* is the one that the flow from each of sources m_{1i} and m_{2i} is equal to or less than s_i, $1 \leqq i \leqq k$, and the flow into sink J_j is exactly p_j, $1 \leqq j \leqq n$.

3.3.2 Algorithm for a feasible schedule

We develop an algorithm which constructs a feasible schedule whenever such one exists.

Let $F(e_i, e_j)$ denote the flow through arc (e_i, e_j). Further, we also use $F(\cdot_i, J_j)$ to denote the total time length during which job J_j can be processed in the interval I_i, where the unit flow corresponds to the unit time length.

Algoritm FS

Step 1. Reduce a given scheduling problem to a corresponding network flow problem and then find a feasible flow. If there exists such flow, then go to Step 2. Otherwise stop. In this case, there exists no feasible schedule.

Step 2. Construct a schedule for the time interval I_i, $1 \leqq i \leqq k$, as follows.

(1) Find some job $J_{h(i)}$ such that

$$F_i = \sum_{j=1}^{n} F(m_{1i}, J_j) + \sum_{j=1}^{h(i)-1} F(v_i, J_j) \leqq s_i$$

and

$$F_i + F(v_i, J_{h(i)}) > s_i,$$

where $1 \leqq h(i) \leqq n$.

(2) Processing on machine M_1: For $1 \leqq j \leqq n$, process job J_j during the time length $F(m_{1i}, J_j)$. Next, for $1 \leqq j \leqq h(i)-1$, process job J_j during $F(v_i, J_j)$. Last, process job $J_{h(i)}$ during $s_i - F_i$.

(3) Processing on M_2: First, process job $J_{h(i)}$ during the time length $F(v_i, J_{h(i)}) + F_i - s_i$. Next, for $h(i)+1 \leqq j \leqq n$, process job J_j during $F(v_i, J_j)$. Last, for $1 \leqq j \leqq n$, process job J_j during $F(m_{2i}, J_j)$. The processing order of jobs on M_1 and M_2 is arbitrary except for job $J_{h(i)}$, which must be processed last on M_1 and first on M_2.

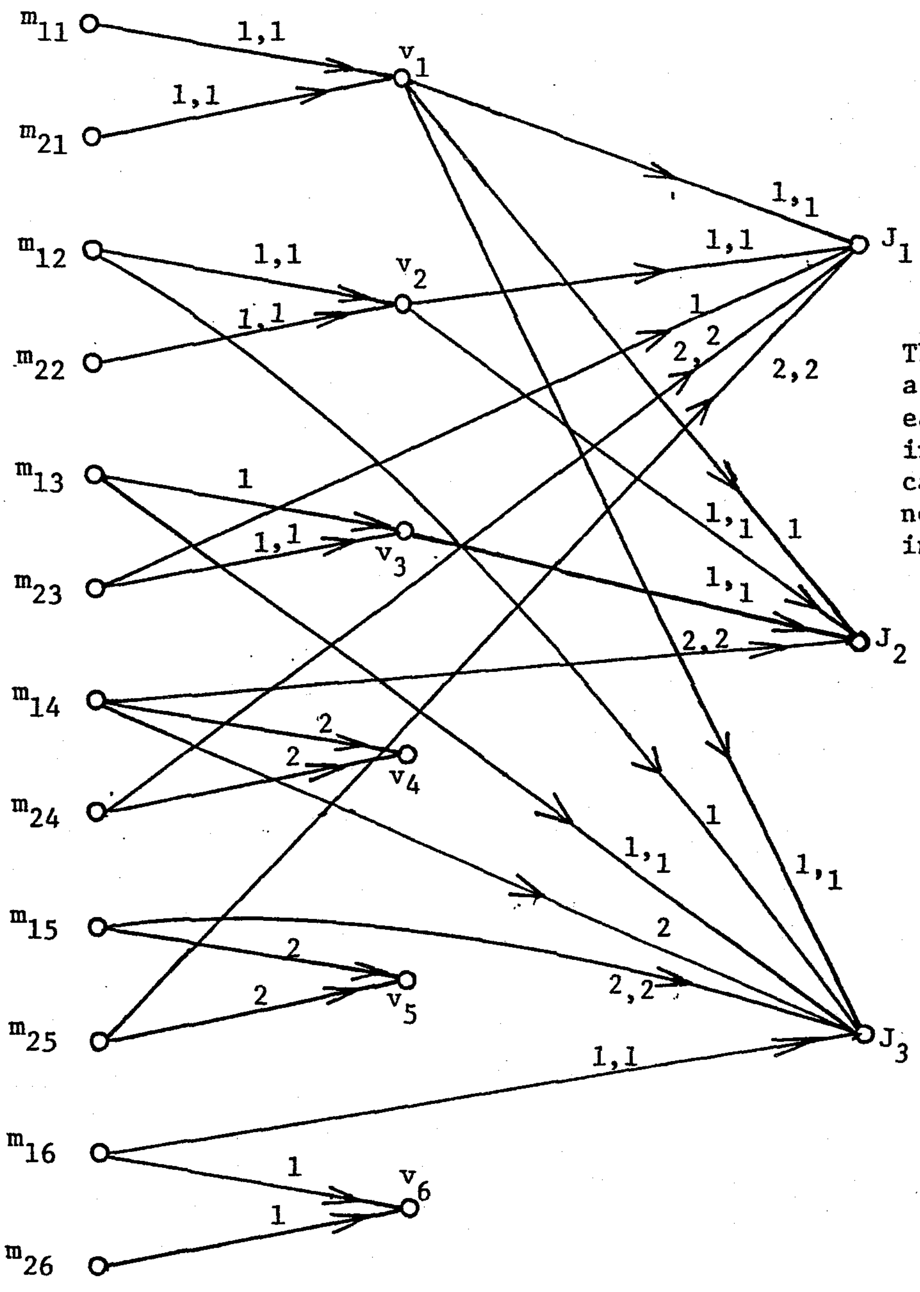
Step 3. Iterate Step 2 for each time interval I_i.

Example 3.1. We consider the following scheduling problem.

$$J = \{J_1, J_2, J_3\}$$
$$(p_1, d_{11}, d_{12}) = (6,2,7)$$
$$(p_2, d_{21}, d_{22}) = (4,5,3)$$
$$(p_3, d_{31}, d_{32}) = (5,8,1)$$

Then, since $D_1=1$, $D_2=2$, $D_3=3$, $D_4=5$, $D_5=7$ and $D_6=8$, we have six time intervals, $I_1=[0,1]$, $I_2=[1,2]$, $I_3=[2,3]$, $I_4=[3,5]$, $I_5=[5,7]$, $I_6=[7,8]$. We can construct the corresponding network as Fig. 3.4.

For the resulting network, the nonzero flow values related to the constructon of the actual schedule are $F(v_1,J_1)=1$, $F(v_1,J_3)=1$, $F(v_2,J_1)=1$, $F(v_2,J_2)=1$, $F(v_3,J_2)=1$, $F(m_{13},J_3)=1$, $F(m_{24},J_1)=2$ $F(m_{14},J_2)=2$, $F(m_{15},J_3)=2$, $F(m_{25},J_2)=2$ and $F(m_{16},J_3)=1$. Thus, there exists a feasible flow as in Fig. 3.4 and we can obtain the feasible schedule as Fig. 3.5.



The numbers attached to each arc indicate its capacity and nonzero flow in this order.

Fig. 3.4. The reduced network and its feasible flow.

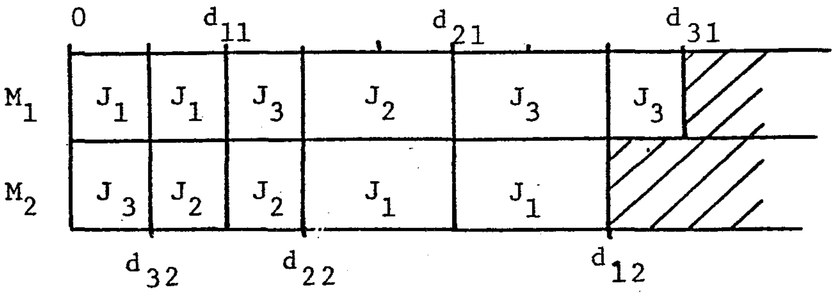


Fig. 3.5. The feasible schedule.

Now, we have the relation between a feasible schedule and a feasible flow.

Theorem 3.3. There exists a feasible schedule if and only if there exists a feasible flow on the reduced network.

Proof. (a) We assume that there exists a feasible schedule. Now, let $t_{ij}(t'_{ij})$ be the time length that job J_j is processed on $M_1(M_2)$ in the time interval I_i. Let the reduced network be $N=(V,E)$, where V is the set of vertices and E is the set of arcs. Then, at most one arc among (m_{1i},J_j), (m_{2i},J_j) and (v_i,J_j) for each sink (job) J_j and interval I_i belongs to E for $1\leqq j\leqq n$ and $1\leqq i\leqq k$. We define the flow through each arc(ϵE) as follows.

(i) When $(m_{1i},J_j)\in E$, we set at

$$F(m_{1i},J_j)=t_{ij}.$$

Note that in this case $t'_{ij}=0$.

(ii) When $(m_{2i},J_j)\in E$, we set at

$$F(m_{2i},J_j)=t'_{ij}.$$

Note that in this case $t_{ij}=0$.

(iii) When $(v_i, J_j) \in E$, we set at

$$F(v_i, J_j) = t_{ij} + t'_{ij},$$

$$F(m_{1i}, v_i) = \sum_{(v_i, J_j)\in E} t_{ij},$$

and

$$F(m_{2i}, v_i) = \sum_{(v_i, J_j)\in E} t'_{ij}.$$

Since there exists a feasible schedule, for $1 \leq i \leq k$ and $1 \leq j \leq n$, we have

$$p_j = \sum_{i=1}^{k} (t_{ij} + t'_{ij}),$$

$$\sum_{j=1}^{n} t_{ij} \leq D_i - D_{i-1},$$

$$\sum_{j=1}^{n} t'_{ij} \leq D_i - D_{i-1},$$

and

$$t_{ij} + t'_{ij} \leq D_i - D_{i-1}.$$

Since s_i, which is a capacity of arc $(\cdot_i, \cdot)$, is equal to $D_i - D_{i-1}$, the flow through each arc does not exceed its capacity. And the flow into sink J_j for $1 \leq j \leq n$ is $\sum_{i=1}^{k} (F(m_{1i}, J_j) + F(m_{2i}, J_j) + F(v_i, J_j)) = \sum_{i=1}^{k} (t_{ij} + t'_{ij}) = p_j$. Thus the demand of each sink is satisfied exactly. Further, the flows from sources m_{1i} and m_{2i} for $1 \leq i \leq k$ are

$$\sum_{j=1}^{n} (F(m_{1i}, J_j) + F(m_{1i}, v_i)) = \sum_{j=1}^{n} t_{ij} \leq s_i,$$

$$\sum_{j=1}^{n} (F(m_{2i}, J_j) + F(m_{2i}, v_i)) = \sum_{j=1}^{n} t'_{ij} \leq s_i.$$

Thus, the flow from each source is not more than the possible supply value s_i. Consequently, we prove that whenever there exists a feasible schedule, there exists a feasible flow in the reduced network.

(b) We assume that there exists a feasible flow. Then, we can show that our algorithm FS constructs a feasible schedule.

In the case $D_i > d_{j1}$, neither (m_{1i}, J_j) or (v_i, J_j) belongs to the arc set E, and thus $F(m_{1i}, J_j) = F(v_i, J_j) = 0$. Similarly, if $D_i > d_{j2}$, then $F(m_{2i}, J_j) = F(v_i, J_j) = 0$. Thus, it is clear that by algorithm FS no job is assigned to unavailable time intervals, i.e., intervals after its due dates. And, the flow into each sink J_j is p_j for $1 \leqq j \leqq n$, from the existence of a feasible flow. Therefore it is sufficient to prove the validity of our algorithm for each interval I_i, $1 \leqq i \leqq k$.

For the time interval I_i, let $T_1 (T_2)$ be the amount of busy periods assigned to $M_1 (M_2)$ by our algorithm. Then, if algorithm FS finds a job $J_{h(i)}$ at Step 2-(1), we have

$$\begin{aligned} T_1 = & \sum_{j=1}^{n} F(m_{1i}, J_j) + \sum_{j=1}^{h(i)-1} F(v_i, J_j) + s_i \\ & - \left(\sum_{j=1}^{n} F(m_{1i}, J_j) + \sum_{j=1}^{h(i)-1} F(v_i, J_j) \right) \\ & = s_i = D_i - D_{i-1}. \end{aligned}$$

On the other hand, if a job $J_{h(i)}$ can not be found by algorithm FS, it is clear that $T_1 \leqq s_i$. Thus, we get

$$\begin{aligned} T_2 = & F(v_i, J_{h(i)}) - s_i + \left(\sum_{j=1}^{n} F(m_{1i}, J_j) + \sum_{j=1}^{h(i)-1} F(v_i, J_j) \right) \\ & + \sum_{j=h(i)+1}^{n} F(v_i, J_j) + \sum_{j=1}^{n} F(m_{2i}, J_j) \\ & = \sum_{j=1}^{n} (F(m_{1i}, J_j) + F(v_i, J_j) + F(m_{2i}, J_j)) - s_i. \end{aligned}$$

Because the first three terms of the right hand side are the sum of flows from sources m_{1i} and m_{2i}, it holds that

$$\sum_{j=1}^{n} (F(m_{1i},J_j)+F(v_i,J_j)+F(m_{2i},J_j)) \leqq 2s_i.$$

Thus we have $T_2 \leqq s_i$. Further, in the interval I_i, the only one job to be processed on both machines is job $J_{h(i)}$, and $F(v_i,J_{h(i)}) \leqq s_i$ from the capacity constraint. Similarly, we can prove the validity of algorithm FS in any other time intervals. Thus the theorem has proved. □

<u>Remark</u>. If network flow problem with $|V|$ vertices is solved by any algorithm with computational time $O(f(|V|))$, our present scheduling problem can be solved in $O(n\log n+kn+f(3k+n))$ time, where n is the number of jobs and k is the number of distinct due dates, for the following reasons.

(i) Sorting the due dates requires $O(n\log n)$ time.

(ii) The reduced network consists of $3k+n$ vertices, i.e., 2k source vertices, k intermediate vertices and n sink vertices.

(iii) Since scheduling the jobs in each of k time intervals requires $O(n)$ time, we require $O(kn)$ time to obtain the whole schedule.

3.3.3 Minimizing maximum lateness

In the last subsection, we proposed an algorithm to construct a feasible schedule if such one exists. We desire to minimize the maximum lateness. In the original probelm, if there exists no feasible schedule, we construct a new problem with $p_i'=p_i$, $d_{i1}'=d_{i1}+L$ and $d_{i2}'=d_{i2}+L$ for $1 \leqq i \leqq n$, where L is some positive constant, i.e., a problem with prolonged due dates. Let L* be the minimum value of L such that there exists a feasible schedule for the new problem. Then it is clear that L* becomes the minimum value of maximum lateness for the original problem. Since for a fixed L, algorithm FS can construct a feasible schedule whenever there exists such one and the possible range of L is $0 \leqq L \leqq \sum_{i=1}^{n} p_i$, we can show, by applying a binary search technique, that an optimal algorithm has a computational time with $O(g(n) \cdot \log \sum_{i=1}^{n} p_i)$, where g(n) is the computational time of algorithm FS.

3.4 $n|m|QI|C_{max}$ Scheduling Problem

In this section, we deal with $n|m|QI|C_{max}$ scheduling problem. A set of jobs $J=\{J_1,J_2,\cdots,J_n\}$ is to be processed on a set of m quasi-identical parallel type machines $M=\{M_1,M_2,\cdots,M_m\}$. Unlike the problems dealt with in Sections 3.2 and 3.3, each job J_i is not always processed on any machine. Now let $I=\{1,2,\cdots,m\}$ be the index set of machines. Job J_i can be processed only on a subset of machines $Q_i=\{M_j|j\in I_i\}$, where $I_i=\{h_i(1),\cdots,h_i(k_i)\}(\subseteq I)$ is an index subset and $0\leqq k_i\leqq m$. Processing time of job J_i on machine M_j is

$$p_{ij}=\begin{cases} p_i & \text{for } M_j\in Q_i \\ \infty & \text{for } M_j\notin Q_i. \end{cases}$$

Our objective is to minimize the maximum completion time. If nonpreemptive schedule is desired, this problem belongs to a class of NP-complete problems. But if preemption is admitted, this problem is tractable. For nonpreemptive csae, we propose a solvable case, in which each job J_i has a unit processing time on Q_i, that is,

$$p_{ij}=\begin{cases} 1 & \text{for } M_j\in Q_i \\ \infty & \text{for } M_j\notin Q_i \end{cases}$$

For preemptive case, on the other hand, each job J_i has an equal but arbitrary processing time on Q_i

On the conventional multi-parallel-machine scheduling problems such as the problems dealt with in Sections 3.2 and 3.3, we assume that each job can be processed on any machine. In the real situations, however, it may happen that each machine can not always process all of given jobs, though the potential capability of machines is equal. For example, a computer program which is executable on some computer can not always be executed on the other same type computers because of the difference of their additional operating systems and so on. Strictly speaking, the machines as mentioned above are not identical but identical in the sense that the capability of machines for executable jobs is identical. So we call them quasi-identical parallel type machines

In the following subsections, we propose an efficient algorithm to construct a feasible schedule for each of nonpreemptive unit processing time case and preemptive arbitrary processing time case. A feasible schedule is defined as follows.

Feasible Schedule: Given a time limit $D(\geq 0)$, a *feasible schedule* is the one that all jobs are completed by the time D.

In next subsection, we show how to reduce the problem obtaining a feasible schedule of nonpreemptive unit processing time case to a maximum cardinality matching problem on a bipartite graph, and develop an efficient algorithm to construct a feasible schedule. Then we show how to minimize the maximum completion time. Similarly, in Subsection 3.3.2, we first show how to reduce the problem obtaining a feasible schedule of preemptive arbitrary processing time case to a network flow problem, and present an efficient algorithm to generate a feasible schedule. Then we show how to minimize the maximum completion time.

3.4.1 Nonpreemptive unit processing time schedule

In this subsection, we assume that each job J_i has a unit processing time $p_i=1$ on Q_i and is to be processed nonpreemptively on Q_i. Our objective is to minimize the maximum completion time. Given an arbitrary time limit D, we propose an algorithm generating a feasible schedule whenever there exists such one. Since

$p_i=1$ for each job J_i, without loss of generality we may assume that the time limit D is integer. For the above purpose, we exploit the solution of maximum cardinality matching problem on a bipartite graph $B=(X,Y,E)$, which is constructed as follows.

Construction of Bipartite Graph

$X=\{v_j(k)\mid j\in I,\ 1\leq k\leq D\}$: vertex set corresponding to machines and time limit.

$Y=\{v(i)\mid i=1,2,\cdots,n\}$: vertex set corresponding to jobs.

$E=\{(v_j(k),v(i))\mid j\in I_i,\ 1\leq k\leq D\}$: edge set connecting vertices $v_j(k)$ and $v(i)$ if machine M_j can process job J_i.

Example 3.2. Let $n=4$, $m=3$, $D=3$, $Q_1=\{M_1,M_2\}$, $Q_2=\{M_2,M_3\}$, $Q_3=\{M_3\}$ and $Q_4=\{M_1\}$. Then we have

$$X=\{v_1(1),\ v_1(2),\ v_1(3),\ v_2(1),\ v_2(2),\ V_2(3),\ v_3(1),\ v_3(2),\ V_3(3)\},$$

$$Y=\{v(1),\ v(2),\ v(3),\ v(4)\},$$

$$\begin{aligned}E=\{&(v_1(1),v(1)),\ (v_1(2),v(1)),\ (v_1(3),v(1)),\\ &(v_2(1),v(1)),\ (v_2(2),v(1)),\ (v_2(3),v(1)),\\ &(v_2(1),v(2)),\ (v_2(2),v(2)),\ (v_2(3),v(2)),\\ &(v_3(1),v(2)),\ (v_3(2),v(2)),\ (v_3(3),v(2)),\\ &(v_3(1),v(3)),\ (v_3(2),v(3)),\ (v_3(3),v(3)),\\ &(v_1(1),v(4)),\ (v_1(2),v(4)),\ (v_1(3),v(4))\}.\end{aligned}$$

The resulting bipartite graph is illustrated in Fig. 3.6.

It is clear that the size of matching on the bipartite graph is at most n. Next, we present an algorithm constructing a feasible schedule from the solution of maximum cardinality matching problem on a bipartite graph.

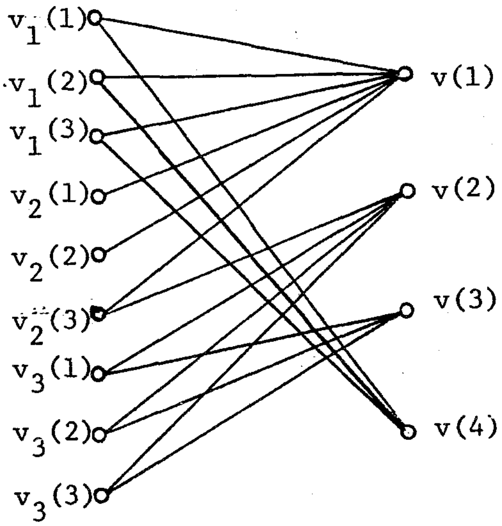


Fig. 3.6. The resulting bipartite graph for Example 3.2.

Algorithm Feasible-I (F-I)

Step 1. Construct a bipartite graph corresponding to the scheduling problem and find a maximum cardinarity matching on that bipartite graph. If the obtained matching has size n, then go to Step 2. Otherwise stop. In such a case, there exists no feasible schedule.

Step 2. If edge $(v_j(k), v(i))$ belongs to the matching obtained in Step 1, process job J_i on machine M_j. And, the processing order of jobs assigned to each machine is arbitrary.

The following theorem shows that whenever there exists a matching of cardinality n, the above algorithm F-I constructs a feasible schedule.

Theorem 3.4. Given an integral time limit D, there exists a feasible schedule if and only if there exists a matching of cardinality n on a bipartite graph as constructed above.

Proof. (i) We first assume that there exists a matching of cardinality n. Then it is sufficient to show that algorithm F-I constructs the feasible schedule. For that purpose, we must show that in a schedule constructed by algorithm F-I, every job is assigned to a suitable machine and the processing of jobs is finished by time D.

Now since edge $(v_j(k),v(i))$ does not belong to E if $M_j \not\in Q_i$, no job is assigned to the unexecutable machine. Also since $|Y|=n$, all jobs are assigned to a suitable machine. Further, since there exist at most D vertices corresponding to an index of each machine, Step 2 assignes at most D jobs to each machine. Thus algorithm F-I can construct a feasible schedule, whenever there exists the matching of cardinality n.

(ii) Next, we assume that there exists a feasible schedule. Then let the jobs processed on machine M_j be $J_{j_1},\cdots,J_{j_{k_j}}$ according to the processing order. Since there exists a feasible schedule, we have $1 \leqq k_j \leqq D$ and $\sum_{j=1}^{m} k_j = n$. Now, as the member of matching we set the edges $(v_j(1),v(j_1)),\cdots,(v_j(k_j),v(j_{k_j}))$ for $1 \leqq j \leqq m$. Then we have the desired matching, that is, the matching of cardinality n. Thus there exists the matching of cardinality n, whenever there exists a feasible schedule. []

As mentioned above, given some integral time limit D, we can construct a feasible schedule whenever there exists such one. Next, we must construct an optimal schedule, completion time of which is a minimum value of time limits when a feasible schedule exists. Similar to the last section, we can find out such time limit by applying a binary search method. To get an optimal schedule, it is sufficient to solve maximum cardinality matching problem at most $log_2 n$ times or to iterate Step 1 in algorithm F-I at most $log_2 n$ times, since $0 \leqq D \leqq \sum_{i=1}^{n} p_i = n$. (Note that it is not necessary to iterate Step 2 in algorithm F-I $log_2 n$ times by virtue of Theorem 3.4.) Here, the matching problem can be solved in polynomial time [18]. Further, the time complexity of Step 2 is O(n). Thus we can construct the optimal schedule in $O(log_2 n \cdot f(n(m+1))+n)$ time, where f(x) is the computational time to obtain a maximum cardinality matching for any bipartite graph with x vertices. (Of cource, f(x) is a polynomial as is already known.)

3.4.2 Preemptive general processing time schedule

We assume that each job J_i has an equal but arbitrary processing time on Q_i. Further, preemptions are admitted. Our objective is to minimize the maximum completion time again. First, we show how to reduce the problem of obtaining a feasible schedule to a network flow problem. The reduced network N=(V,E), where V is a vertex set consisting of two disjoint subsets S and T, and E is a directed arc set, is constructed as follows.

<u>Construction of Reduced Network</u>

(i) $S=\{s_j | j=1,2,\cdots,m\}$: a set of source vertices, in which each source has a maximum possible amount of supply D.

(ii) $T=\{t_i | i=1,2,\cdots,n\}$: a set of sink vertices, in which each sink t_i has a demand p_i

(iii) $E=\{(s_j,t_i) | M_j \in Q_i,\ 1 \leqq j \leqq m,\ 1 \leqq i \leqq n\}$: a set of directed arcs, in which each arc is directed from s_j to t_i and has a capacity D.

Note that a source vertex s_j corresponds to a machine M_j and a sink vertex t_i to a job J_i. Also, we may assume that $D \geq \max_{1 \leq i \leq n} p_i$, since our present objective is to obtain a feasible schedule.

Example 3.3. Let n=4, m=3, D=5, $Q_1=\{M_1,M_2\}$, $Q_2=\{M_2\}$, $Q_3=\{M_3\}$, $Q_4=\{M_1,M_2,M_3\}$, $p_1=3$, $p_2=1$, $p_3=3$ and $p_4=2$. See Fig. 3.7.

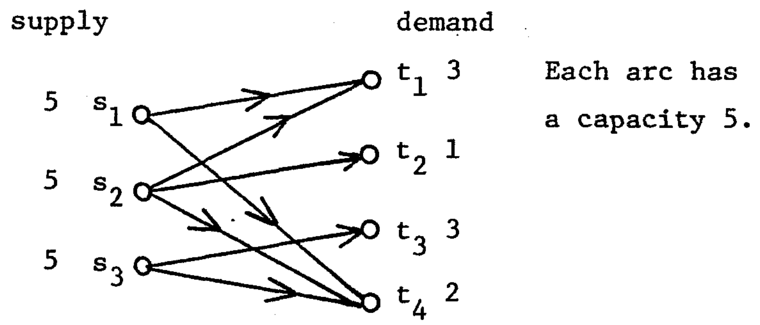


Fig. 3.7. The reduced network for Example 3.3.

We define a feasible flow in the above network as follows.

Feasible Flow: A *feasible flow* is the following,

(i) the flow from each source is no more than D,

(ii) the flow into each sink t_i is exactly p_i,

(iii) the flow through each arc is at most D.

Next, when there exists a feasible flow, we construct a solution of $n|m|O|C_{max}$ preemptive scheduling problem based on the feasible flow on the above network. This problem can be solved efficiently by Gonzalez and Sahni algorithm (G-S algorithm) [5]. Our algorithm exploits the schedule generated by G-S algorithm to obtain the feasible schedule of our problem. Let $F(s_j,t_i)$ be the amount of flow on arc (s_j,t_i), and

$J'=\{J_1',\cdots,J_n'\}$: a set of open shop type jobs,

$M'=\{M_1',\cdots,M_m'\}$: a set of open shop type machines

O_{ij}: operation of job J_i' to be processed on machine M_j',

$p_{ij}=F(s_j,t_i)$: processing time of operation O_{ij}.

For $n|m|O|C_{max}$ preemptive schedule, Gonzalez and Sahni also showed that the maximum completion time C^*_{max} of the schedule generated by their algorithm meets the lower bound showed in Section 3.3, that is,

$$C^*_{max} = \max(\max_j \sum_{i=1}^{n} p_{ij},\ \max_i \sum_{j=1}^{m} p_{ij}).$$

Algorithm Feasible II (F-II)

Step 1. Construct the reduced network and find a feasible flow. If there exists no feasible flow, then stop. (In such a case, there exists no feasible schedule.) Else, go to Step 2.

Step 2. Let $F(s_j,t_i)$ be the amount of flow through arc (s_j,t_i) in the obtained feasible flow. Based on this flow value, construct a corresponding $n|m|O|C_{max}$ preemptive scheduling problem and solve that problem.

Step 3. Replace machine M_j' and operation O_{ij} in the above scheduling problem with machine M_j and job J_i in an original scheduling problem, respectively. This schedule is the desired feasible schedule.

In the following theorem, we prove that the existence of feasible flow is equivalent to that of feasible schedule through algorithm F-II.

Theorem 3.5. Given a time limit D, there exists a feasible schedule if and only if there exists a feasible flow.

Proof. (i) We assume that there exists a feasible flow. Then, we must show that algorithm F-II always constructs a feasible schedule. Now, since no operation of job J_i' in open shop problem is processed at the same time, no job in our original problem is processed on several machines. It is clear that each machine processes at most one job at the same time. Further, since, by virtue of the construction of our network, arc (s_j, t_i) does not belong to the arc set E of network if $M_j \notin Q_i$, no job is assigned to the nonexecutable machines. Therefore it is left to show that all jobs are completed by time D.

By Gonzalez and Sahni, the maximum completion time, C^*_{max}, of schedule constructed in Step 2 is

$$C^*_{max} = \max(\max_j \sum_{i=1}^{n} p_{ij}, \max_i \sum_{j=1}^{m} p_{ij}).$$

Since $p_{ij}=F(s_j,t_i)$, for each source s_j the total amount of flow out of s_j in the feasible flow is

$$\sum_{i=1}^{n} F(s_j,t_i) = \sum_{i=1}^{n} p_{ij}.$$

Thus we have

$$\max_j \sum_{i=1}^{n} p_{ij} \leqq D,$$

since the possible supply of source s_j is D. Also the total amount of flow into each sink t_i in the feasible flow is

$$\sum_{j=1}^{m} F(s_j,t_i) = \sum_{j=1}^{m} p_{ij}.$$

Then, since in the feasible flow the demand of sink t_i is exactly p_i, we get

$$\max_i \sum_{j=1}^{m} p_{ij} = \max_i p_i.$$

On the other hand, since we assume that $\max_i p_i \leqq D$, we have

$$\max_i \sum_{j=1}^{m} p_{ij} \leqq D.$$

Consequently, we have

$$C^*_{max} \leqq D.$$

(ii) We assume that there exists a feasible schedule. Let p'_{ij} be the amount of processing of job J_i on M_j. Then we fix the flow value through arc (s_j,t_i) at $F(s_j,t_i)=p'_{ij}$. Now since there exists a feasible schedule, we have $F(s_j,t_i)\leqq D$. Thus the capacity constraint for each arc is satisfied. Also we have

$$\sum_{j=1}^{m} p'_{ij} = p_i$$

and

$$\sum_{i=1}^{n} p'_{ij} \leq D.$$

Accordingly our current flow becomes a feasible flow. $\square$

As mentioned above, given a time limit D we can costruct a feasible schedule whenever there exists such one. Next we must construct a schedule to minimize the maximum completion time.

Then similar to the last subsection, we can find out a desired schedule, i.e., optimal schedule, by exploiting a binary search technique. Since the possible ranges of maximum completion time C_{max} and time limit D are $\max_i p_i \leq C_{max}$ (or D) $\leq \sum_{i=1}^{n} p_i$, we can construct an optimal schedule by solving $log_2 \sum_{i=1}^{n} p_i$ network flow problems. Further, since both a network flow problem and $n|m|O|C_{max}$ preemptive scheduling problem are solved in a polynomial time, an optimal schedule can also be constructed in a polynomial time.

CHAPTER 4

SCHEDULING PROBLEMS WITH CHANGEABLE MACHINE SPEED

4.1 Introduction

In this chapter, we extend ordinary scheduling problems with constant machine speed to the cases with changeable speed.

The first one is an extension of $n|m|U|C_{max}$ scheduling problem in which each machine M_j has a constant speed q_j. In the extended problem, each machine speed s_j of machine M_j is a continuous nonnegative variable. Our objective is to determine both the optimal speeds of machines and an optimal preemptive schedule with respect to some cost function f_{max}. Thus, the problem is an $n|m|GU|f_{max}$ preemptive scheduling problem.

The second is an extension of $n|2|MX|C_{max}$ scheduling problem as analyzed in Section 2.4. In this extended problem, again each machine speed s_j is a continuous nonnegative variable. The objective is to determine both the optimal speeds of machines and an optimal nonpreemptive schedule with respect to some cost function f_{max}. This problem is an $n|2|GMX|f_{max}$ nonpreemptive scheduling problem.

In the traditional scheduling problems, each machine has a predetermined machine speed q_j including a unit speed $q_j=1$. For $n|m|U|C_{max}$ preemptive scheduling problem, Gonzalez and Sahni [6] presented a polynomial time algorithm to construct an optimal schedule. Concerning shop type machine, on the other hand, only the problem with unit machine speeds have been analyzed.

In Section 4.2, polynomial time algorithms are presented to find the assignments of optimal speeds to each machine for a variety of cost functions. Further, we show that if we relax some of assumptions, then the resulting problems become NP-hard.

In Section 4.3, we develop a polynomial time solution procedure to determine both the optimal speeds of machines and an optimal schedule for the generalized mixed shop scheduling problem.

4.2 A Generalized Uniform Machine System

We consider a scheduling problem determining both the optimal speeds of machines and an optimal preemptive schedule on parallel type machines.

Most scheduling problems considered in the literature attempt to schedule with a given set of jobs and one or more machines with constant speeds. In this section, we will assume that we are able to determine both the machines available and the schedule. Our assumption is that it is possible to change the machine speeds, and to raise up the speed takes more cost.

This model is reasonable for the real time systems that must complete a given set of jobs within a specified time.

Our model is an extention of $n|m|U|C_{max}$ preemptive scheduling problem. Now, for a given machine speeds, we are able to find an optimal schedule using the uniform machine algorithm of Gonzalez and Sahni [6]. The properties of this algorithm are quite flexible in choosing the optimal machine speeds.

In the following, we give a more formal description of the problem. We consider a *generalized uniform machine* system (GUM system) that has the following properties.

1. There is a set of jobs $J=\{J_1,\cdots,J_n\}$ to be processed and each job J_i has an amount of processing requirement p_i.
2. There is a set of m parallel type machines $M=\{M_1,\cdots,M_m\}$ available, and the speed of each machine is a continuous nonnegative variable. If machine M_j has speed s_j, a cost $f_j(s_j)$ is incurred.
3. Preemptions are allowed.

For a system of machines with speeds $S=(s_1,\cdots,s_m)$, the machine cost is $\sum_{j=1}^{m} f_j(s_j)$. Our objective is to find a vector $(s_1,\cdots,s_m)$ that minimizes $f_{max}=f_0(T)+\sum_{j=1}^{m} f_j(s_j)$, where $f_0(t)$ is a completion cost incurred for finishing the last job at time t and T is the minimum value of maximum completion time for the given speed vector. Thus the problem is an $n|m|GU|f_{max}$ preemptive scheduling problem. We first show how to find a GUM system with minimum machine cost which can complete all jobs by the time D, given a deadline D. This problem is called a *Deadline Problem*. Then we show how to use the solution algorithm for the deadline problem to solve the original problem.

In order to find an optimal solution to these problems, we will make the following assumptions about the machine cost functions $f_1,f_2,\cdots,f_m$:

(i) $f_j(0)=0$.

(ii) $f_j(x)$ is positive and strictly increasing for $x>0$.

(iii) $f_j(x)\leqq f_{j+1}(x)$ for all $j=1,2,\cdots,m-1$ and $x>0$.

(iv) $f_j'(x)$, the derivative of $f_j(x)$, is continuous and increasing for $x>0$.

A set of machines with properties (i)-(iv) will be called an *ordered machine system*.

Intuitively, these restrictions represent a system of machines that are ordered with respect to cost, so that the more fast the machine speed, the more cost incurred. Assumption (iii) implies that there is always an optimal solution with $s_1\geqq s_2\geqq\cdots\geqq s_m$. Without loss of generality, we can assume that the processing requirements are sorted as $p_1\geqq p_2\geqq\cdots\geqq p_n$.

We need not the explicit form of cost functions but we can get the values of

(a) $f_j(x)$ for $j=1,2,\cdots,m$ and $x>0$,

(b) $f_j'(x)$ for $j=1,2,\cdots,m$ and $x>0$,

(c) the solution x of $f_j'(x)=f_{j+1}(y)$ for any given $y>0$ and $j=1,2,\cdots,m-1$.

Once the speed vector is specified, a schedule minimizing the maximum completion time can be found in $O(m\cdot log_2 m+n)$ time using the G-S algorithm. We now briefly review the relationship between the machine speeds and the minimum value of maximum completion times. Horvath et al. [11] have shown that the maximum completion time C_{max} of an optimal preemptive schedule was given as follows.

$$(4.1) \qquad C_{max}=\max\{ \max_{1\leq j\leq m} \{\frac{P_j}{S_j}\}, \frac{P_n}{S_m}\},$$

where $P_j=\sum_{k=1}^{j} p_k$, $j=1,2,\cdots,n$, and $S_j=\sum_{k=1}^{j} s_k$, $j=1,2,\cdots,m$. Thus, the deadline problem is equivalent to the following problem:

$$(4.2) \qquad \text{minimize} \sum_{j=1}^{m} f_j(s_j)$$

$$(4.3) \qquad \text{subject to } S_j\geq P_j/D \qquad j=1.,2,\cdots,m-1$$

$$S_m\geq P_n/D.$$

Any speed vector s that satisfies (4.3) is said to be *feasible*.

In Subsection 4.2.1, we show how to solve the deadline problem using the derivatives of the machine functions. In Subsection 4.2.2, we show how to use the solution algorithm for the deadline problem to solve the $n|m|GU|f_{max}$ preemptive scheduling problem. Subsection 4.2.3 gives a fast implementation for a more restricted class of cost functions. Subsection 4.2.4 discusses an extension of the cost model which includes setup costs. In Subsection 4.2.5, we show that several versions of this problem are NP-hard.

4.2.1 The deadline problem

Our algorithm can construct an optimal speed vector successively. First an optimal speed vector to complete job J_1 by time D is found, and then the algorithm proceeds to an optimal vector to complete J_1 and J_2 by time D. Finally, we will find an optimal vector to complete all jobs by time D. Each speed vector we find will be a lower bound on all future speed vectors. The next speed vector can be obtained successively by increasing the element of precedent one whose marginal cost (the derivative of the cost function evaluated at its current speed) is smallest. We now prove several properties of this solution technique.

Lemma 4.1. For $k<m$, there exists an optimal vector $(s_1,s_2,\cdots,s_m)$ that completes the first k large jobs by time D and has the following three properties:

(i) $s_{k+1}=s_{k+2}=\cdots=s_m=0$,

(ii) $s_k=P_k/D$,

(iii) $f_1'(s_1)\geq f_2'(s_2)\geq\cdots\geq f_k'(s_k)$.

Proof. Property (i) follows the assumption $f_j(x)\leq f_{j+1}(x)$ and the fact that each job can run on at most one machine at the same time.

To prove (ii), we note that if $S_k<P_k/D$, we can not complete all k jobs by time D, and if $S_k>P_k/D$, we can reduce the speed of the slowest machine with nonzero speed and get a new vector that is feasible and has lower cost.

To prove (iii), we note that if $f_j'(s_j)<f_{j+1}'(s_{j+1})$, for some $\varepsilon>0$ we can increase s_j by ε, decrease s_{j+1} by ε, and obtain a new feasible vector of lower cost. $\square$

Corollary 4.1. For $m\leqq k\leqq n$, there exists an optimal solution vector $(s_1,s_2,\cdots,s_m)$ that completes the first k large jobs by time D and has the following two properties:

(ii)' $S_m=P_k/D$,

(iii)' $f_1'(s_1)\geqq f_2'(s_2)\geqq\cdots\geqq f_m'(s_m)$.

In this subsection, we will consider only optimal speed vectors that satisfy properties (i) (ii) and (iii) in Lemma 4.1 and have $s_1\geqq s_2\geqq\cdots\geqq s_m$.

Lemma 4.2. Given an optimal vector $S=(s_1,s_2,\cdots,s_m)$ for the first k large jobs, there exists an optimal vector $\bar{S}=(\bar{s}_1,\bar{s}_2,\cdots,\bar{s}_m)$ for the k+1 large jobs such that $\bar{s}_j\geqq s_j$ for $j=1,2,\cdots,m$.

Proof. Suppose $\bar{S}$ is an optimal vector that violates the lemma. Let j be the least index such that $\bar{s}_j<s_j$. Since $\bar{S}_k\geqq P_k/D=S_k$, there must be some $\bar{s}_l>s_l$ for $l\leqq k$. Let l be the least of such index. Let $\Delta=\min(\bar{s}_l-s_l,\ s_j-\bar{s}_j)$. Let S^* be the vector obtained from $\bar{S}$ by replacing $\bar{s}_j$ by $\bar{s}_j+\Delta$ and replacing $\bar{s}_l$ by $\bar{s}_l-\Delta$. The vector S^* is a feasible solution for the first k+1 large jobs. We now show that the machine cost of S^*, which we write $C(S^*)$, is not greater than the machine cost of $\bar{S}$, which we write $C(\bar{S})$. By the construction of above S^*, we have

$$C(S^*)=C(\bar{S})+(f_j(\bar{s}_j+\Delta)-f_j(\bar{s}_j))-(f_l(\bar{s}_l)-f_l(\bar{s}_l-\Delta)). \tag{4.4}$$

We consider the vector $\hat{S}$ obtained from S by replacing s_j by $s_j-\Delta$ and s_l by $s_l+\Delta$. Note that $\hat{s}_j\geqq\bar{s}_j$ and $\hat{s}_l\leqq\bar{s}_l$. If $l<j$, then $\hat{S}$ is

clearly feasible for the first k jobs. If $l>j$, then since l is the least index with $\bar{s}_l>s_l$, we know that $\hat{s}_i\geq\bar{s}_i$, $i=1,2,\cdots,l-1$. Thus it holds that

$$\hat{S}_i\geq\bar{S}_i\geq P_i/D \quad \text{for } i=1,2,\cdots,l-1,$$

$$\hat{S}_i\geq S_i\geq P_i/D \quad \text{for } i=l,l+1,\cdots,k.$$

Therefore, $\hat{S}$ is feasible for the first k large jobs. Since S is optimal, we must have

(4.5) $$C(\hat{S})-C(S)=(f_l(s_l+\Delta)-f_l(s_l))-(f_j(s_j)-f_j(s_j-\Delta))\geq 0.$$

Since f'_l snd f'_j are increasing and $\bar{s}_l\geq\bar{s}_l+\Delta$, $\bar{s}_j\leq s_j-\Delta=\hat{s}_j$, we have

(4.6) $$f_l(s_l+\Delta)-f_l(s_l)\leq f_l(\bar{s}_l)-f_l(\bar{s}_l-\Delta)$$

and

(4.7) $$f_j(s_j)-f_j(s_j-\Delta)\geq f_j(s_j+\Delta)-f_j(\bar{s}_j).$$

Combining equations (4.5)-(4.7), we have

(4.8) $$(f_l(\bar{s}_l)-f_l(\bar{s}_l-\Delta))-(f_j(\bar{s}_j+\Delta)-f_j(\bar{s}_j))\geq 0$$

From (4.4) and (4.8) we know that $C(S^*)\leq C(\bar{S})$. Thus, by successively applying the transform we used to get S^* from $\bar{S}$, we will obtain an optimal vector that satisfies the lemma. This vector also satisfies properties (i) (ii) and (iii) of Lemma 4.1. $\square$

Lemma 4.3. If S is an optimal vector for jobs with processing requirements $p_1\geq p_2\geq\cdots\geq p_k$ $(k\leq m)$ and $f'_{l-1}(s_{l-1})>f'_l(s_l)=f'_{l+1}(s_{l+1})=\cdots=f'_k(s_k)$, then for any set of values Δ_l, $\Delta_{l+1},\cdots,\Delta_j$ such that

(i) $p_k+D(\Delta_l+\Delta_{l+1}+\cdots+\Delta_k)\leq p_{k-1}$ and

(ii) $f'_{l-1}(s_{l-1})\geq f'_l(s_l+\Delta_l)=f'_{l+1}(s_{l+1}+\Delta_{l+1})=\cdots=f'_k(s_k+\Delta_k)$,

the vector $(s_1,\cdots,s_{l-1},s_l+\Delta_l,\cdots,s_k+\Delta_k)$ is an optimal vector for jobs with processing requirements $p_1, p_2, \cdots, p_{k-1}, p_k + D(\Delta_l + \cdots + \Delta_k)$.

Proof. Let $\hat{S}$ be such that

$$\hat{s}_i = \begin{cases} s_i & \text{for } i=1,2,\cdots,l-1, \\ s_i + \Delta_i & \text{for } i=l, l+1, \cdots, k \end{cases}$$

If $\hat{S}$ is not optimal, then there exists a vector S^* whose cost is less than $\hat{S}$. By lemma 4.2, we can assume that $s_i^* \geq s_i$ for $i=1, \cdots, k$. Thus there must exist indices j and r such that $s_j^* > \hat{s}_j$, $s_r^* < \hat{s}_r$ and $r \geq l$. If $j \geq l$, we have $f_j'(\hat{s}_j) = f_r'(\hat{s}_r)$ and if $j < l$, $f_j'(\hat{s}_j) \geq f_r'(\hat{s}_r)$. Since the derivatives are increasing, it is possible to increase s_r^* and decrease s_j^*, and obtain a new vector that is feasible and has cheaper cost than S^*. This conclusion contradicts our assumption that S^* is optimal. Consequently, no vector can be cheaper than $\hat{S}$. □

We are ready to describe the algorithm for the deadline problem. In this algorithm, we can treat the small n-m+1 jobs as a single job with processing requirement $\bar{p}_m = \sum_{i=m}^{n} p_i$. We also have $\bar{p}_i = p_i$ for $i=1,2,\cdots,m-1$.

Algorithm DL

Step 1. Set $s_1 = \bar{p}_1/D$, $s_i = 0$ for $i=2,\cdots,m$ and $k=2$

Step 2. If $S_k \geq \bar{P}_k/D$, then go to Step 5.

Step 3. Let l be the smallest index with $f_l'(s_l) = f_{l+1}'(s_{l+1}) = \cdots = f_k'(s_k)$ for $l \leq k$. Find values $\Delta_l, \Delta_{l+1}, \cdots, \Delta_k$ such that

$$f_{l-1}'(s_{l-1}) \geq f_l'(s_l + \Delta_l) = f_{l+1}'(s_{l+1} + \Delta_{l+1}) = \cdots = f_k'(s_k + \Delta_k)$$

and

$(\sum_{j=l}^{k}(s_j+\Delta_j)+\sum_{j=1}^{l-1}s_j)=\bar{P}_k/D$. If no such values exists, then find values such taht $f'_{l-1}(s_{l-1})=f'_l(s_l+\Delta_l)=\cdots=f'_k(s_k+\Delta_k)$.

Step 4. Set $s_j=s_j+\Delta_j$ for $j=l,\cdots,k$. Return to Step 2.

Step 5. Return to Step 2 setting k to k+1.

Example 4.1. Let $n=6$, $m=3$, $D=1$, $p_1=10$, $p_2=6$, $p_3=4$, $p_4=2$, $p_5=2$, $p_6=2$, $f_1(x)=x^2$, $f_2(x)=2x^2+4x$ and $f_3(x)=3x^2+6x$. Then, we have $\bar{p}_1=10$, $\bar{p}_2=6$, $\bar{p}_3=10$, $f'_1(x)=2x$, $f'_2(x)=4x+4$ and $f'_3(x)=6x+6$. Figure 4.1 gives a sample execution of algorithm DL for this example.

The computational time of algorithm DL is dominated by Step 3. Since the value of l in Step 3 must decrease with execution of Step 3 except for the last. Step 3 is executed at most $O(m)$ times for each k. If the values Δ_l, $\Delta_{l+1},\cdots,\Delta_k$ can be computed in time $O(d)$, where d depends on the types of actual cost functions, the total running time is $O(m(m+d))$.

Theorem 4.1. The algorithm DL computes a minimum cost vector which complete jobs with processing requirements p_1, p_2, $\cdots$, p_n by time D.

Proof. It is clear that $S_k \geq \bar{P}_k/D$ for $k=1,2,\cdots,m$. So the vector is feasible. Consequently, the initial vector constructed in Step 1 is optimal for J_1. By Lemmas 4.2 and 4.3, Steps 2, 3 and 4 produce a vector that is optimal for the first k large jobs. □

Variables	s_1	s_2	s_3	l	Δ_1	Δ_2	Δ_3	Comments
Initial values	10	—	—	—	—	—		
k=2	10	4	—	2	—	4	—	$f_1'(10)=20=f_2'(4)$
	34/3	14/3	—	1	4/3	2/3	—	$f_1'(34/3)=68/3=f_2'(14/3)$
k=3	34/3	14/3	25/9	3	—	—	(25/9)	$f_3'(25/9)=68/3$
	(168/11)	73/11	45/11	1	130/33	65/33	130/33	$f_1'(168/11)=336/11=$ $f_2'(73/11)=f_3'(45/11)$

Figure 4.1. A sample execution of DL.

4.2.2 General solution method for the $n|m|GU|f_{max}$ preemptive scheduling problem

For any fixed maximum completion time, T, we can use the algorithm DL to find a minimum cost vector that completes all jobs by T. If we decrease the completion time to T-Δ, we can reduce the completion cost by $f_0(T)-f_0(T-\Delta)$, but the optimal vector to complete all jobs by T-Δ is more expensive. Now, we will compare the completion cost and the machine cost. We assume that the completion cost $f_0(t)$ has a derivative which is continuous and nondecreasing for t>0. We show that the magnitude of the change in the cost of speed vector is decreasing in T.

We define

$$F(t) \triangleq \text{cost of an optimal speed vector for } D=t$$

Let S be an optimal vector for maximum completion time T. We have

$$S_i \geq P_i/T \quad \text{for } i=1,2,\cdots,m-1,$$

and

$$S_m = P_n/T$$

In optimal vector $\bar{S}$ for maximum completion time T-ε, we have

$$\bar{S}_i \geq P_i/(T-\varepsilon) \quad \text{for } i=1,2,\cdots,m-1,$$

and

$$\bar{S}_m = P_n/(T-\varepsilon).$$

Lemma 4.4. For any t and ε such that $\varepsilon>0$ and $t-2\varepsilon>0$, it holds that

$$F(t-2\varepsilon)-F(t-\varepsilon)>F(t-\varepsilon)-F(t).$$

Proof. Let s, $\bar{s}$, and $\hat{s}$ be optimal vectors for deadlines of t, $t-\varepsilon$ and $t-2\varepsilon$. Without loss of generality, we may assume that s, $\bar{s}$ and $\hat{s}$ are vectors constructed by the algorithm DL. We know that $\hat{s}_i \geq \bar{s}_i \geq s_i$ for $i=1,2,\cdots,m$.

Let $\bar{s}_i = s_i + \Delta_i$ and $\hat{s}_i = \bar{s}_i + \bar{\Delta}_i$ for $i=1,2,\cdots,m$. Further, let $i_1 < i_2 < \cdots < i_k < m$ be the indices such that

$$\bar{S}_{i_j} = P_{i_j}/(t-\varepsilon).$$

Because of the properties of the algorithm DL, we have

1. $f_1'(\bar{s}_1) = f_2'(\bar{s}_2) = \cdots = f_{i_1}'(\bar{s}_{i_1})$,

2. $f_{i_1+1}'(\bar{s}_{i_1+1}) = f_{i_1+2}'(\bar{s}_{i_1+2}) = \cdots = f_{i_2}'(\bar{s}_{i_2})$,

$\vdots$

k. $f_{i_{k-1}+1}'(\bar{s}_{i_{k-1}+1}) = \cdots = f_{i_k}'(\bar{s}_{i_k})$,

k+1. $f_{i_k+1}'(\bar{s}_{i_k+1}) = \cdots = f_m'(\bar{s}_m)$.

The properties 1 through k+1, and the fact that the derivatives are increasing, imply that

$$\begin{aligned} F(t-\varepsilon)-F(t) < {} & (\Delta_1+\Delta_2+\cdots+\Delta_{i_1}) f_{i_1}'(\bar{s}_{i_1}) \\ & + (\Delta_{i_1+1}+\cdots+\Delta_{i_2}) f_{i_2}'(\bar{s}_{i_2}) \\ & \vdots \\ & + (\Delta_{i_k+1}+\cdots+\Delta_m) f_m'(\bar{s}_m), \end{aligned} \tag{4.9}$$

and

(4.10) $$F(t-2\varepsilon)-F(t-\varepsilon) > (\bar{\Delta}_1+\bar{\Delta}_2+\cdots+\bar{\Delta}_{i_1})f'_{i_1}(\bar{s}_{i_1})$$
$$+(\bar{\Delta}_{i_1+1}+\cdots+\bar{\Delta}_{i_2})f'_{i_2}(\bar{s}_{i_2})$$
$$\vdots$$
$$+(\bar{\Delta}_{i_k+1}+\cdots+\bar{\Delta}_m)f'_m(\bar{s}_m).$$

Now we have

$$(\Delta_1+\Delta_2+\cdots+\Delta_{i_1})=P_{i_1}/(t-\varepsilon)-S_{i_1}$$
$$\leqq P_{i_1}/(t-\varepsilon) - P_{i_1}/t$$
$$= P_{i_1}/t\cdot(t-\varepsilon)$$

and

$$(\bar{\Delta}_1+\bar{\Delta}_2+\cdots+\bar{\Delta}_{i_1})\geqq P_{i_1}/(t-2\varepsilon) - P_{i_1}/(t-\varepsilon)$$
$$> P_{i_1}/t\cdot(t-\varepsilon).$$

Thus it holds that $\Delta_1+\cdots+\Delta_{i_1}<\bar{\Delta}_1+\cdots+\bar{\Delta}_{i_1}$. In a similar way, we show that

(4.11) $$\sum_{l=1}^{i_j}\Delta_l < \sum_{l=1}^{i_j}\bar{\Delta}_l \quad \text{for } j=1,2,\cdots,k \text{ and}$$
$$\sum_{l=1}^{m}\Delta_l < \sum_{l=1}^{m}\bar{\Delta}_l.$$

Thus, since $f'_{i_1}(\bar{s}_{i_1})\geqq f'_{i_2}(\bar{s}_{i_2})\geqq\cdots\geqq f'_m(\bar{s}_m)$, from (4.9), (4.10) and (4.11) we have

$$F(t-2\varepsilon)-F(t-\varepsilon) > F(t-\varepsilon)-F(t). \qquad \square$$

Since $F(t)=\sum_{i=1}^{m} f_i(\bar{s}_i)$, $\sum_{k=1}^{i_j} \bar{s}_k=P_{i_j}/t$ for $j=1,2,\cdots,k$ and $\sum_{k=1}^{m} \bar{s}_k=P_n/t$, from the properties 1 through k+1 in above lemma, we have

$$F'(t)=-\frac{1}{t^2}[P_{i_1}f'_{i_1}(\bar{s}_{i_1})+\cdots+(P_n-P_{i_k})f'_m(\bar{s}_m)].$$

And by this lemma, $F'(t)$ is decreasing. Thus, since the derivative of f_0 is nondecreasing, the unique value of D which minimize $f_0(D)+F(D)$ is the solution of $f'_0(D)=-F'(D)$. (Note that $f_0(0)=0$ and $F(0)=\infty$.)

4.2.3 A special class of cost functions

We consider machines with costs

$$f_j(x)=c_j x^k \quad \text{for } j=1,2,\cdots,m \text{ with } c_1\leqq c_2\leqq\cdots\leqq c_m,$$

where $k(\geqq 1)$ is a constant. We first give a fast implementation for the deadline problem and then show how to find the minimum (optimum) value of maximum completion times.

In the algorithm DL, we repeatedly found a group of machines that had the same marginal cost, and increased the speeds of all the machines. We take advantage of the fact that, if for any intermediate speed vector two machines have the same marginal cost, they have the same marginal cost in the final solution. Using this property, we combine all machines with the same marginal cost into a single *composite machine*. The speed of the composite machine is the sum of the speeds of its individual machines, but the marginal cost is the same as its individual machines. In the following lemma, we show how to form an appropriate cost function for a composite machine.

Lemma 4.5. If two machines have cost functions $f_1(x)=c_1x^k$ and $f_2(x)=c_2x^k$, and the speeds s_1 and s_2 such that $f_1'(s_1)=f_2'(s_2)$ are assigned to each machine, then we have

$$[kc_1\ c_2(s_1+s_2)^{k-1}]/(c_1^{1/(k-1)}+c_2^{1/(k-1)})^{k-1}=f_1'(s_1)$$
$$=f_2'(s_2).$$

Proof. By the assumption, we have $f_1'(s_1)=kc_1s_1^{k-1}=kc_2s_2^{k-1}$. Solving for c_1, we get $c_1=c_2(s_2/s_1)^{k-1}$. Substituting this value into the left hand side of the equation of this lemma, we obtain

$$kc_2^2\left(\frac{s_2}{s_1}\right)^{k-1}\left(\frac{s_1+s_2}{(s_2/s_1)c_2^{1/(k-1)}+c_2^{1/(k-1)}}\right)^{k-1}$$
$$=kc_2\left(\frac{s_2}{s_1}\cdot\frac{s_1+s_2}{(s_2/s_1)+1}\right)^{k-1}$$
$$=kc_2s_2^{k-1}=f_2'(s_2)=f_1'(s_1). \qquad \square$$

Note that if we set

$$C=\frac{c_1c_2}{(c_1^{1/(k-1)}+c_2^{1/(k-1)})^{k-1}}, \tag{4.12}$$

then we have $kC(s_1+s_2)^{k-1}=f'(s_1+s_2)$, where

$$f(x)=Cx^k. \tag{4.13}$$

Thus we can replace any two machines with a single composite machine whose coefficient is as in (4.12). As long as the speed of the composite machine is the sum of the speeds of the individual machines, its marginal cost is the same as that of the individual machine. Since the composite machine has the same type of cost function as the original machines, formula (4.12) can be applied to any member of machines and the resulting machines still have the same marginal cost.

We are now ready to describe the algorithm. The algorithm produces a vector of composite machines. Associated with the i-th composite machine are

(i) L_i= list of indices of the original machines that were combined;

(ii) $\bar{c}_i$= coefficient of its cost function;

(iii) $\bar{s}_i$= sum of the speed of all the combined original machines.

Thus all machines in the list L_i have their marginal cost $k\bar{c}_i\bar{s}_i^{k-1}$. And if machine M_j is in this composite machine, we find its speed $s_j=\bar{s}_i(\bar{c}_i/c_j)^{1/(k-1)}$ by solving

$$(4.14)\qquad kc_js_j^{k-1}=k\bar{c}_i\bar{s}_i^{k-1}.$$

After processing the first l large jobs, the algorithm constructs a list of composite machines that corresponds to an optimal solution for this l jobs. The algorithm again forms a new list for the $l+1$ large jobs as follows. Let $\bar{s}_j$ for $j=1,2,\cdots,i$ be the speeds of the i composite machines forming an optimal solution for the l large jobs, and let $\bar{c}_j$ for $j=1,2,\cdots,i$ be the coefficients of their cost functions. We initially assume that $(i+1)_{th}$ composite machine is the original machine M_{l+1} with speed

$\bar{s}_{i+1}=\frac{p_{l+1}}{D}$, and cost coefficient $\bar{c}_{i+1}=c_{l+1}$. This vector is optimal for the $l+1$ large jobs, if

$$(4.15) \qquad kc_{l+1}\bar{s}_{i+1}^{-k-1} \leq k\bar{c}_i\bar{s}_i^{-k-1}.$$

If the condition (4.15) does not hold, then machine M_{l+1} and all machines in composite machine i have the same marginal cost in the optimal vector. Thus we can merge composite machines i and i+1. Further, we compare the new marginal cost of composite machine i with the marginal cost of i-1. We continue the merging process untill we have j composite machines and the marginal cost of composite machine j is not greater than the marginal cost of j-1.

In the following, we present a formal description of the algorithm. We again assume that the n-m+1 small jobs are merged so that

$$\bar{p}_j=p_j \qquad \text{for } j=1,2,\cdots,m-1$$

$$\bar{p}_m=p_m+p_{m+1}+\cdots+p_n.$$

<u>Algorithm MC</u>

Step 1. Set $\bar{s}_0=\infty$, $\bar{c}_o=\infty$, $\bar{c}_1=c_1$, $\bar{s}_1=P_1/D$, $L_1=\{M_1\}$, i=1 and $l=2$.

Step 2. Update i, L_i, $\bar{c}_i$ and $\bar{s}_i$ as follows.

(1) i=i+1,

(2) $L_i=\{l\}$,

(3) $\bar{c}_i=c_l$,

(4) $\bar{s}_i=\bar{p}_l/D$.

Step 3. If $\bar{c}_i\bar{s}_i^{-k-1} > \bar{c}_{i-1}\bar{s}_{i-1}^{-k-1}$, then go to Step 4. Else go to Step 5.

Step 4. Update $\bar{c}_{i-1}$, $\bar{s}_{i-1}$, L_{i-1} and i as follows

(1) $\bar{c}_{i-1}=(\bar{c}_i\bar{c}_{i-1})/(\bar{c}_i^{1/(k-1)}+\bar{c}_{i-1}^{1/(k-1)})^{k-1}$,

(2) $\bar{s}_{i-1}=\bar{s}_i+\bar{s}_{i-1}$,

(3) $L_{i-1}=L_{i-1}\cup L_i$,

(4) $i=i-1$.

Return to Step 3.

Step 5. If $l \geqq m$, then stop. Otherwise, return to Step 2 setting l to $l+1$.

The computational time of algorithm MC is dominated by the loop of Steps 3 and 4. Each execution of the loop decrements i. We increment i by one (m-1) times in Step 2-(1), and i can not become smaller than one. Thus, this loop is executed at most (m-1) times. Thus the loop takes time O(m) if we count each of the numerical operations and the set operations as a unit time. Further all other steps in the algorithm can be taken with time O(m), and thus the actual speeds of the original machines can also be computed with time O(m). On the other hand, to find the first m large processing requirements and sort them in advance, we require time O(n) and time $O(m\,log_2 m)$, respectively. Consequently, the total time to construct an optiaml schedule is $O(m\,log_2 m+n)$, since the actual schedule is constructed in time $O(m\,log_2 m+n)$ by using the G-S algorithm.

The validity of algorithm MC follows from Lemma 4.5 and the fact that if the condition in Step 3 is satisfied, all the machines in composite machine i and i-1 must have the same marginal cost in an optimal vector.

Now, we show how to solve the original problem for this class of cost functions. In the algorithm MC, each machine speed is propotional to D^{-1} and the comparison of Step 3 does not depend on D. Thus the same composite machines are always formed regardless of the values of D. Therefore the optimal speed can be represented as

$$s_j=u_j/D \quad \text{for } j=1,2, \quad ,m,$$

where u_j is the optimal speed when D=1. Moreover, the total machine cost is

$$\sum_{j=1}^{m} c_j(u_j/D)^k=U/D^k,$$

where $U=\sum_{j=1}^{m} c_j u_j^k$. Then the total cost f_{max} is

$$(4.16) \qquad f_{max}=f_0(t)+U/t^k.$$

With regard to cost functions f_0, it is easy to find a t that minimizes f_{max}. Especially, regarding the simplest cost function $f_0(t)=c_0t$ and k=2, the optimal solution has

$$t=\left(\frac{2U}{c_0}\right)^{1/3}.$$

4.2.4 Including setup costs

Often it is useful to consider machine cost functions of the form

$$g_i(x)=\begin{cases} v_i+f_i(x) & x>0, \\ 0 & x=0, \end{cases}$$

where $f_i(x)$ has the property given in the beginning of this section. Thus v_i is a fixed setup cost incurred by using machine M_i.

If $v_1 \leq v_2 \leq \cdots \leq v_m$, then we can solve the problem as follows. The optimal speeds have $s_1 \geq s_2 \geq \cdots \geq s_k > s_{k+1} = \cdots = s_m = 0$ for some $k \leq m$. If k is fixed, the total setup cost is always $\sum_{j=1}^{k} v_j$. Thus the setup costs are ignored. Therefore the problem including the setup costs is reduced to the original problems with k machines. Then an optimal vector is found with m calls for the algorithm DL.

4.2.5 NP-hardness

We show that if we relax some of our assumptions about the cost functions, the resulting problems become NP-hard. Informally, a problem, whether a member of NP or not, is NP-hard if we can transform an NP-complete problem to it and it can not be solved in polynomial time unless P=NP. Thus, in an intuitive sense, the NP-hard problems are at least as hard as the NP-complete problems. For the formal definition of NP-hard, refer to Garey and Johnson [4]. We first consider arbitrary setup costs and then machines with discrete speeds. We use the NP-complete theory for a *Subset Sum* problem [4] defined as follows.

Subset Sum Problem: Given $S=\{a_1,a_2,\cdots,a_n\}$ and b, where $a_1,\cdots,a_n$ and b are integers, is there a subset $\bar{S} \subseteq S$ such that

$$\sum_{a_i \in \bar{S}} a_i = b \text{ ?}$$

We show that if we have cost functions of the form $f_i(x)=v_i+c_ix^2$ for $x>0$ and $f_i(0)=0$, then the problem to find a minimum cost solution for a given deadline is NP-hard. Given a solution for the subset sum problem, we can construct a solution of the deadline problem as follows: We make n+k jobs with processing requirements,

$$p_1=p_2=\cdots=p_n=\lfloor b/n \rfloor,$$

$$p_{n+1}=\cdots=p_{n+k}=1, \quad \text{where } k=b-n\lfloor b/n \rfloor.$$

Thus we have $P_{n+k}=b$. Also there are n machines whose cost functions are

$$v_i=a_i,$$

$$c_i=1/a_i \quad \text{for } i=1,2,\cdots,n.$$

In this case, the deadline is D=1.

Lemma 4.6. There exists a solution for the subset sum problem if and only if the deadline problem has a solution with cost 2b.

Proof. (i) Suppose that there is a solution $\bar{S}$ for the subset sum problem. Then the speed vector with $s_i=a_i$ if $a_i \in \bar{S}$ and $s_i=0$ if $a_i \notin \bar{S}$ has cost

$$\sum_{a_i \in \bar{S}} \left(a_i + \frac{1}{a_i}(a_i)^2\right)=2b.$$

From the construction of the processing requirements, any solution $S_n \geq b$ is feasible, where $S_n=\sum_{j=1}^{n} s_j$. Therefore the deadline problem has a solution with cost 2b.

(ii) Suppose that there exists a solution S with cost for a deadline problem. Then we know that $S_n=b$, and show that $s_i>0$ implies $s_i=a_i$. Concerning the machines with nonzero speeds, we define

$$r_i \triangleq \frac{(a_i+s_i^2/a_i)}{s_i} = \frac{a_i}{s_i} + \frac{s_i}{a_i}.$$

Now we have

$$\frac{a_i}{s_i} + \frac{s_i}{a_i} = 2 \quad \text{if } a_i=s_i$$

and

$$\frac{a_i}{s_i} + \frac{s_i}{a_i} > 2 \quad \text{if } a_i \neq s_i.$$

Since the overall ratio of cost to speed is $(2b/S_n)=2$, each r_i must have value 2. Thus each nonzero speed s_i has value a_i, and so the nonzero s_i's form a solution to the subset sum problem. ▯

Theorem 4.2. The deadline problem with arbitrary setup costs is NP-hard.

Proof. Lemma 4.6 shows that we can a subset sum problem by a transform into a corresponding deadline problem. The transformation can be achieved polynomially. Therefore we have proved the theorem. ▯

Next we consider a discrete version of our machine system in which each machine M_i can take only two possible speeds 0 or q_i. We show that the *discrete deadline problem* is NP-hard even for the following cost functions:

$$f_i(x)=c_i x^2 \quad \text{for } i=1,2,\cdots,m.$$

Again, we start with a subset sum problem and construct a solution for the discrete deadline problem with n+k jobs and n machines as follows:

$p_1=p_2=\cdots=p_n=\lfloor b/n \rfloor$,

$p_{n+1}=\cdots=p_{n+k}=1$, where $k=b-n\lfloor b/n \rfloor$

$q_i=a_i$ for $i=1,2,\cdots,n$

$c_i=\dfrac{1}{a_i}$ for $i=1,2,\cdots,n$

$D=1$.

Lemma 4.7. There is a solution to the discrete deadline problem with cost b if and only if the subset sum problem has a solution.

Proof. Similar to Lemma 4.6. $\square$

Theorem 4.3. The discrete version of the deadline problem is NP-hard.

Proof. Analogous to Theorem 4.2. $\square$

Corollary 4.2. The discrete $n|m|GU|f_{max}$ scheduling problem is NP-hard.

Proof. Given a solution of subset sum problem, convert it to a corresponding discrete deadline problem as in Lemma 4.6. Instead of a deadline, we use a completion cost $f_0(t)=bt$ and total cost $bt+S_m \geq bt+P_{n+k}/t=b(t+1/t)$. This cost function is minimized when there is a solution to subset sum problem and $t=1$. Thus completion cost has a cost 2b if and only if subset sum problem has a solution. $\square$

4.3 Generalized Mixed Shop Scheduling

In this section, we consider an extension of $n|2|MX|C_{max}$ nonpreemptive scheduling problem to the changeable speed case. This problem is specified as follows.

(1) There is a set of n jobs $J=\{1,2,\cdots,n\}$ to be processed on two machines M_1 and M_2.

(2) Each job i consists of two operations, one of which is to be processed on M_1 and the other on M_2.

(3) The job set J consists of two disjoint subsets F and O. F is a set of flow shop type jobs and O is a set of open shop type jobs.

(4) A speed of each machine is a variable. Processing requirements of each job i on M_1 and M_2 are a_i and b_i, respectively.

(5) No preemption is allowed.

(6) The objective is to determine an optimal speed of each machine and an optimal schedule to minimize the total cost f_{max} associated with the maximum completion time and the speeds of machines.

This is an $n|2|GMX|f_{max}$ nonpreemptive scheduling problem.

In this problem, the actual schedule can be constructed by the algorithm for the ordinaly $n|2|MX|C_{max}$ nonpreemptive scheduling problem discussed in Section 2.4. So we can focus on obtaining the optimal speeds.

In Subsection 4.3.1, we formulate the main problem P. The problem P can be divided into two subproblems $\bar{P}$ and $\bar{\bar{P}}$. In order to solve $\bar{P}$, we introduce auxiliary (or supplementary) problems. Similarly for $\bar{\bar{P}}$, supplementary problems are introduced. In Subsection 4.3.2, we develop a polynomial time solution procedure for the main problem P and clarify its time complexity.

4.3.1 Formulation of the problem

Let s_1 and s_2 be the speeds of machines M_1 and M_2, respectively. Then the processing times of job i become a_i/s_1 on M_1 and b_i/s_2 on M_2. Further, let C_{max} be the maximum completion time of an optimal schedule as the function of the machine speeds.

The following problem P is the main problem considered in this section.

$$\text{P:}\quad \text{Minimize}\quad f_{max}=c_0 C_{max}^{q_1}+c_1 s_1^{q_2}+c_2 s_2^{q_2}$$

$$\text{subject to}\quad s_1, s_2>0,$$

where c_0, c_1 and c_2 are positive constants and , q_1 and q_2 are positive integers. The problem P is divided into subproblems $\bar{P}$ and $\bar{\bar{P}}$ as follows.

$$\bar{P}:\quad \text{Minimize}\quad c_0 C_{max}^{q_1}+c_1 s_1^{q_2}+c_2 s_2^{q_2}$$

$$\text{subject to}\quad A_F/s_1 \geq B_0/s_2 \text{ and } s_1, s_2>0,$$

$$\text{where } A_F=\sum_{i\in F} a_i \text{ and } B_0=\sum_{i\in 0} b_i.$$

$$\bar{\bar{P}}:\quad \text{Minimize}\quad c_0 C_{max}^{q_1}+c_1 s_1^{q_2}+c_2 s_2^{q_2}$$

$$\text{subject to}\quad A_F/s_1 < B_0/s_2 \text{ and } s_1, s_2>0.$$

Note that $\bar{P}$ corresponds to Case 1 in Section 2.4 and $\bar{\bar{P}}$ to other cases. Thus in the problem $\bar{P}$ we have

$$C_{max}=\max(CF^*,\ (A_F+A_0)/s_1,\ (B_F+B_0)/s_2), \tag{4.19}$$

where $A_0=\sum_{i\in 0} a_i$, $B_F=\sum_{i\in F} b_i$ and CF* is the maximum completion time of an optimal schedule when only jobs in F are considered subject to machine speeds s_1 and s_2.

An optimal schedule giving CF* is determined by the following binary transitive rule R_0, which is the variation of Johnson's rule.

R_0: If $\min(s_1'a_j, s_2'b_k) \leqq \min(s_1'a_k, s_2'b_j)$, where $s_1'=1/s_1$ and $s_2'=1/s_2$, then the processing of job j precedes that of job k.

R_0 is equivalent to the following relation R, since s_1' and s_2' are strictly positive.

R: If $\min(\gamma a_j, b_k) \leqq \min(\gamma a_k, b_j)$, then the processing of job j precedes that of job k, where $\gamma = s_1'/s_2'$.

The relation R implies that the candidate points of γ, where an optimal schedule changes, are $\gamma_{jk}=b_k/a_j$ for $j,k \in F$, where if $a_j=0$, then γ_{jk} is set to ∞. Considering finite $\gamma_{jk} \geqq B_0/A_F$ and sorting the different γ_{jk}'s in an increasing order, let

$$\gamma_0 \overset{\Delta}{=} B_0/A_F < \gamma_1 < \gamma_2 < \cdots < \gamma_p < \gamma_{p+1} \overset{\Delta}{=} M,$$

where M is a sufficiently large number and p is the cardinality of different γ_{jk}'s. Note that $1 \leqq p \leqq n_1^2$, where $n_1 = |F|$.

Theorem 4.4. If we have

(4.20) $\quad \min(\bar{\gamma} a_j, b_k) \leqq \min(\bar{\gamma} a_k, b_j) \quad$ for $\gamma_i < \bar{\gamma} < \gamma_{i+1}$,

(4.21) $\quad \min(\tilde{\gamma} a_j, b_k) \leqq \min(\tilde{\gamma} a_k, b_j) \quad$ for $\gamma_i \leqq \tilde{\gamma} \leqq \gamma_{i+1}$

holds.

Proof. First note that the following cases are possible.

Case 1. $\bar{\gamma} a_j \leqq b_k$ and $\bar{\gamma} a_k \leqq b_j$

Case 2. $\bar{\gamma} a_j > b_k$ and $\bar{\gamma} a_k \leqq b_j$

Case 3. $\bar{\gamma}a_j \leqq b_k$ and $\bar{\gamma}a_k > b_j$

Case 4. $\bar{\gamma}a_j > b_k$ and $\bar{\gamma}a_k > b_j$

Case 1. $\bar{\gamma}a_j \leqq b_k$ and $\bar{\gamma}a_k \leqq b_j$.

From (4.20), it holds that

$$(4.22) \qquad \min(\bar{\gamma}a_j, b_k) = \bar{\gamma}a_j \leqq \min(\bar{\gamma}a_k, b_j) = \bar{\gamma}a_k \text{ or } a_j \leqq a_k.$$

From definition of γ_i, γ_{i+1} and the assumption, we have

$$\tilde{\gamma} \leqq \gamma_{i+1} \leqq \min(b_k/a_j, b_j/a_k).$$

Thus $\min(\tilde{\gamma}a_j, b_k) = \tilde{\gamma}a_j$ and $\min(\tilde{\gamma}a_k, b_j) = \tilde{\gamma}a_k$ hold. Combination of (4.20) with (4.22) shows that $\min(\tilde{\gamma}a_j, b_k) = \tilde{\gamma}a_j \leqq \tilde{\gamma}a_k = \min(\tilde{\gamma}a_k, b_j)$, that is, we have (4.21).

Proofs of other cases can be done in the same way as in this case, and so it is omitted. $\square$

Theorem 4.4 means that an optimal schedule for some $\gamma \in (\gamma_i, \gamma_{i+1})$ is also optimal for any $\gamma \in [\gamma_i, \gamma_{i+1}]$. Accordingly, CF* can be expressed on the interval $[\gamma_i, \gamma_{i+1}]$ as follows.

$$CF^* = s_2' \max_{1 \leqq j \leqq n_1} \left(\gamma' \sum_{k=1}^{j} a_{[k]} + \sum_{k=j}^{n_1} b_{[k]}\right),$$

where $\gamma' = (\gamma_i + \gamma_{i+1})/2$ and [k] denotes the k-th job index corresponding to this γ'.

4.3.2 Solution procedure for subproblem $\bar{P}$

From the expression of CF*, the feasible region of $\bar{P}$, $\{(s_1', s_2') | s_1', s_2' > 0,\ \gamma = s_1'/s_2' \geqq B_0/A_F\}$ is divided into the subregions $\{(s_1', s_2') | s_1', s_2' > 0,\ \gamma \in [\gamma_i, \gamma_{i+1}]\}$ for $i = 1, 2, \cdots, p$. Each subregion must be divided further as follows.

First, on the interval $[\gamma_i,\gamma_{i+1}]$ we consider (n_1+2) linear functions y_i of γ,

$$y_i=\gamma A_i+B_i, \quad \text{for } i=1,2,\cdots,n_1+2,$$

where

$$A_i=\begin{cases} \sum_{k=1}^{i} a_{[k]} & \text{for } i=1,2,\cdots,n_1, \\ A_F+A_0 & \text{for } i=n_1+1, \\ 0 & \text{for } i=n_1+2, \end{cases}$$

$$B_i=\begin{cases} \sum_{k=i}^{n_1} b_{[k]} & \text{for } i=1,2,\cdots,n_1, \\ 0 & \text{for } i=n_1+1, \\ B_F+B_0 & \text{for } i=n_1+2, \end{cases}$$

$\gamma\in[\gamma_i,\gamma_{i+1}]$ and $[k]$ denotes the k-th job index of F in an optimal schedule corresponding to $\gamma'=(\gamma_i+\gamma_{i+1})/2$. Let y be the function defined by the maximum value of y_i's for each γ, i.e., $y=\max_{1\leqq i\leqq n_1+2}(y_i)$ if using a suppressed notation. By utilizing Megiddo's algorithm [24], y can be determined in at most $O(n_1 \log n_1)$ time, and y is a piecewise linear increasing convex function. Arranging the breaking points of y in an increasing order, we have

$$\gamma_i^0=\gamma_i<\gamma_i^1<\cdots<\gamma_i^h<\cdots<\gamma_i^{m_i}<\gamma_i^{m_i+1}=\gamma_{i+1},$$

where $1\leqq m_i\leqq n_1$.

Now we introduce the following subproblems $\bar{P}_i^h$ of $\bar{P}$ for $h=0, 1,\cdots,m_i$ and $i=0,1,\cdots,p$.

$\bar{P}_i^h$: Minimize $\bar{C}_i^h = c_0(s_1'A_\alpha + s_2'B_\alpha)^{q_1} + c_1 s_1^{q_2} + c_2 s_2^{q_2}$

subject to $\gamma = s_1'/s_2'$ $[\gamma_i^h, \gamma_i^{h+1}]$ and $s_1, s_2 > 0$,

where α is the index of y_α that gives y on the subinterval $[\gamma_i^h, \gamma_i^{h+1}]$.

By solving all $\bar{P}_i^h$'s and choosing the best solution among optimal solutions of $\bar{P}_i^h$, $\bar{P}$ can be solved. Therefore an optimal speeds and an optimal schedule can be found.

4.3.3 Solution procedure for $\bar{P}_i^h$

By the famous inequality between arithmetic and geometric means, it holds that

$$\bar{C}_i^h = c_0(s_1'A_\alpha + s_2'B_\alpha)^{q_1} + c_1 s_1^{q_2} + c_2 s_2^{q_2}$$
$$= c_0 s_2'^{q_1}(\gamma A_\alpha + B_\alpha)^{q_1} + s_2^{q_2}\{c_1(1/\gamma)^{q_2} + c_2\}$$
$$\geqq (q_1+q_2)[(c_0/q_2)^{q_2}(\gamma A_\alpha + B_\alpha)^{q_1 q_2}(1/q_1)^{q_1}\{c_1(1/\gamma)^{q_2} + c_2\}^{q_1}]^{1/(q_1+q_2)},$$

where the equality occurs if and only if

$$s_2' = \left(\frac{q_2}{c_0 q_1} \cdot \frac{c_1\gamma^{-q_2} + c_2}{(\gamma A_\alpha + B_\alpha)^{q_1}}\right)^{1/(q_1+q_2)}$$

Thus, in order to solve $\bar{P}_i^h$, it is sufficient to find a minimizer γ_i^{h*} of

$$f(\gamma) = (\gamma A_\alpha + B_\alpha)^{q_1 q_2}(c_1\gamma^{-q_2} + c_2)^{q_1}$$

on the interval $[\gamma_i^h, \gamma_i^{h+1}]$. Once γ_i^{h*} is found, an optimal solution (s_{1i}^h, s_{2i}^h) of $\bar{P}_i^h$ is constructed as follows.

$$s'^{h}_{2i}=\left(\frac{q_2}{c_0q_1}\cdot\frac{c_1(\gamma_i^{h*})^{-q_2}+c_2}{(\gamma_i^{h*}A_\alpha+B_\alpha)^{q_1}}\right)^{1/(q_1+q_2)}$$

$$s'^{h}_{1i}=\gamma_i^{h*}s'^{h}_{2i}.$$

Differentiating $f(\gamma)$ with respect to γ, we have

$$f'(\gamma)=q_1q_2A_\alpha c_2(\gamma A_\alpha+B_\alpha)^{q_1q_2-1}(c_1\gamma^{-q_2}+c_2)^{q_1-1}\gamma^{-(q_2+1)}$$

$$\times\{\gamma^{q_2+1}-(B_\alpha c_1/A_\alpha c_2)\}.$$

Since $f'(\gamma)$ changes its sign at most once, γ_i^{h*} is determined as follows.

(i) If $(\gamma_i^{h})^{q_2+1}\geqq(B_\alpha c_1)/(A_\alpha c_2)$, then $\gamma_i^{h*}=\gamma_i^{h}$.

(ii) If $(\gamma_i^{h+1})^{q_2+1}\leqq(B_\alpha c_1)/(A_\alpha c_2)$, then $\gamma_i^{h*}=\gamma_i^{h+1}$.

(iii) If $(\gamma_i^{h})^{q_2+1}<(B_\alpha c_1)/(A_\alpha c_2)<(\gamma_i^{h+1})^{q_2+1}$, then $\gamma_i^{h*}=$

$$\left(\frac{B_\alpha c_1}{A_\alpha c_2}\right)^{-1/(q_2+1)}$$

In order to solve $\bar{P}$, we must compute

$$\bar{C}^{h*}_{i*}(s'^{h*}_{1i*},s'^{h*}_{2i*})=\min_{i,h}(\bar{C}^{h}_{i}(s'^{h}_{1i},s'^{h}_{2i})).$$

Then for $\bar{P}$, machine speeds $\bar{s}_1$ and $\bar{s}_2$ are determined as $1/s'^{h*}_{1i*}$ and $1/s'^{h*}_{2i*}$, respectively and an optimal schedule is constructed by applying the algorithm in Section 2.4, where the processing times of job j are $a_j/\bar{s}_1$ on M_1 and $b_j/\bar{s}_2$ on M_2.

4.3.4 Solution procedure for subproblem $\bar{\bar{P}}$

From the results of Section 2.4, $s_1'A_F \leq s_2'B_0$ implies

$$C_{max}=\max(s_1'(A_F+A_0),\ s_2'(B_F+B_0),\ \max_{i\in O}(s_1'a_i+s_2'b_i))$$

$$=s_2'\max(\gamma(A_F+A_0),\ B_F+B_0,\ \max_{i\in O}(\gamma a_i+b_i)).$$

Now we define the (n_2+2) linear functions of γ as follows, where $n_2=|O|$.

$$z_j=\gamma\bar{\bar{A}}_j+\bar{\bar{B}}_j \quad \text{for } j=1,2,\cdots,n_2+2,$$

where $\bar{\bar{A}}=a_j$, $\bar{\bar{B}}_j=b_j$ for $j=1,2,\cdots,n_2$ corresponding to job $j\in O$, and $\bar{\bar{A}}_{n_2+1}=A_F+A_0$, $\bar{\bar{B}}_{n_2+1}=0$, $\bar{\bar{A}}_{n_2+2}=0$ and $\bar{\bar{B}}_{n_2+2}=B_F+B_0$. Then if we define

$$z=\max_{1\leq i\leq n_2+2}(z_i),$$

we have $C_{max}=s_2'z$. This z is just same form as y, and can be obtained by Meggido's algorithm in at most $O(n_2 \log n_2)$ computational time. Arranging the breaking points of z on the interval $(0,B_0/A_F]$ in an increasing order, we get the sequence

$$\gamma_0'=\varepsilon<\gamma_1'<\ \cdots\ <\gamma_{p'}'<\gamma_{p'+1}'=B_0/A_F, \text{ where}$$

ε is a sufficiently small positive value and p' is the number of breaking points on $(0,B_0/A_F]$. Note that for $\gamma\in[\gamma_i',\gamma_{i+1}']$, we have $z=z_\beta$ for a certain β, $1\leq\beta\leq n_2+2$. Then the following subproblems $\bar{\bar{P}}_i$ of $\bar{\bar{P}}$ for $i=0,1,\cdots,n_2+2$ are introduced.

$$\bar{\bar{P}}_i: \quad \text{Minimize} \quad \bar{\bar{C}}^i = c_0(s_1'\bar{\bar{A}}_\beta + s_2'\bar{\bar{B}}_\beta)^{q_1} + c_1 s_1^{q_2} + c_2 s_2^{q_2}$$

$$\text{subject to } \gamma = (s_1'/s_2') \in [\gamma_i', \gamma_{i+1}'],\ s_1', s_2' > 0,$$

where β is the subscript of z_β that gives z on this interval. Again, solving all $\bar{\bar{P}}_i$ and choosing the best solution among optimal solutions of $\bar{\bar{P}}_i$, $\bar{\bar{P}}$ can be solved, i.e., each optimal speed and an optimal schedule can be found. Solution procedure for $\bar{\bar{P}}_i$ is quite same as that for $\bar{P}_i^h$ and so it is omitted.

Now, we denote a minimal solution of $\bar{\bar{C}}^i$ with (s_{1i}', s_{2i}') by

$$\bar{\bar{C}}^{i*}(s_{1i*}', s_{2i*}') = \min_{1 \leq i \leq p'} (\bar{\bar{C}}^i(s_{1i}', s_{2i}')).$$

Then optimal speeds $\bar{\bar{s}}_1$ and $\bar{\bar{s}}_2$ are determined $1/s_{1i*}'$ and $1/s_{2i*}'$, respectively. Further, the corresponding optimal schedule can be found by solving the ordinary $n|2|MX|C_{max}$ nonpreemtive scheduling problem with processing times $a_j/\bar{\bar{s}}_1$ and $b_j/\bar{\bar{s}}_2$ for $j \in O$.

4.3.5 Solution procedure for the main problem P

It is clear that the optimal speeds s_1^* and s_2^* of the main problem P can be found by comparing $\bar{C}_{i*}^{h*}(s_{1i*}'^{h*}, s_{2i*}'^{h*})$ with $\bar{\bar{C}}^{i*}(s_{1i*}', s_{2i*}')$. Using s_1^* and s_2^*, an optimal schedule can be found by the algorithm in Section 2.4, where the processing times of job j are a_j/s_1^* on M_1 and b_j/s_2^* on M_2.

Theorem 4.5. The above solution procedure finds the optimal speeds of M_1 and M_2, and an optimal schedule in at most $O(n^3 \log_2 n)$ computational time for given q_1 and q_2, if any power and root can be computed on O(1) time.

Proof. The validity of our procedure is proved already from the preceding discussions. Therefore we show only the complexity of our procedure.

The computation of γ_i takes $O(n_1^2 \mathit{log} n_1)$ time, since the number of γ_{jk} is at most $O(n_1^2)$ and sorting $O(n_1^2)$ elements takes $O(n_1^2 \mathit{log} n_1)$ time. Next, an optimal schedule of jobs in F on some interval $[\gamma_i, \gamma_{i+1}]$ is determined in $O(n_1 \mathit{log} n_1)$ time. Once an optimal order is determined, then y can be obtained in $O(n_1 \mathit{log} n_1)$ time. Thus $(s'^{h*}_{1i*}, s'^{h*}_{2i*})$ can be found in $O(n_1^3 \mathit{log} n_1)$ time.

Similarly its complexity for $\bar{\bar{P}}$ is $O(n_2 \mathit{log} n_2)$, since p' is at most $O(n_2)$. Finally an optimal schedule can be constructed in $O(n \mathit{log} n)$ time. Consequently, the complexity of our solution procedure for the main problem P is $O(n^3 \mathit{log} n)$. $\square$

References

[1] A.V.Aho, J.E.Hopcroft and J.D.Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass. (1974).

[2] R.Berger, The Undecidability of the Domino Problem (Mem. American Math. Soc., No.66), *American Mathematical Society Providence*, RI.(1966).

[3] J.Edmonds, Path, Trees and Flowers, *Canadian J. Math.*, Vol. 17, 449-467(1965).

[4] M.R.Garey and D.S.Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.H.Freeman and Company, San Francisco(1979).

[5] T.Gonzalez and S.Sahni, Open Shop Scheduling to Minimize Finish Time, *J. Assoc. Comput. Mach.*, Vol.23, 665-679(1976).

[6] T.Gonzalez and S.Sahni, Preemptive Scheduling of Uniform Processor Systems, *J. Assoc. Comput. Mach.*, Vol.25, 92-101 (1978).

[7] R.L.Graham, Bounds on Certain Multiprocessing Anomalies, *Bell System Tech. J.*, Vol.45, 1563-1581(1966).

[8] R.L.Graham, Bounds on Multiprocessing Timing Anomalies, *SIAM J. Appl. Math.*, Vol.17, 263-269(1969).

[9] R.L.Graham, E.L.Lawler, J.K.Lenstra and A.H.G.Rinnooy Kan, Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey, *Annals of Discrete Math.*, Vol.5, 287-326(1979).

[10] W.A.Horn, Some Simple Scheduling Problems, *Naval Res. Logist. Quart.*, Vol.21, 177-185(1974).

[11] E.C.Horvath, S.Lam and R.Sethi, A Level Algorithms for Preemptive Scheduling, *J. Assoc. Comput. Mach.*, Vol.24, 32-43 (1977).

[12] H.Ishii, C.Martel, T.Masuda and T.Nishida, A Generalized Uniform Processor System, *Operations Res.*, Vol.33, 346-362 (1985).

[13] J.R.Jackson, Scheduling a Production Line to Minimize Maximum Lateness, *Research Report 43, Management Science Project*, Univ. of California, Los Angels(1955).

[14] J.R.Jackson, An Extension of Johnson's Results on Job Lot Scheduling, *Naval Res. Logist. Quart.*, Vol.3, 201-203(1956).

[15] S.M.Johnson, Optimal Two- and Three-Stage Production Schedules with Setup Times Included, *Naval Res. Logist. Quart.*, Vol.1, 61-68(1954).

[16] H.Kise, H.Ibaraki and H.Mine, Performance Analysis of Six Approximation Algorithms for the One-Machine Maximum Lateness Scheduling Problem with Ready Times, *J. Operations Res. Soc. Japan*, Vol.22, 205-224(1979).

[17] E.L.Lawler, Optimal Sequencing of a Single Machine Subject to Precedence Constraints, *Management Sci.*, Vol.19, 544-546(1973).

[18] E.L.Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York(1976).

[19] T.Masuda, H.Ishii and T.Nishida, Some Bounds on Approximation Algorithms for $n/m/I/L_{max}$ and $n/2/F/L_{max}$ Scheduling Problems, *J. Operations Res. Soc. Japan*, Vol.26, 212-224 (1983).

[20] T.Masuda, H.Ishii and T.Nishida, Two-Machine Scheduling Problem for Jobs with Generalized Due Dates, *Math. Japonica*, Vol.30, 117-125(1985).

[21] T.Masuda, H.Ishii and T.Nishida, The Mixed Shop Scheduling Problem, *Discrete Appl. Math.*, Vol.11 175-186(1985).

[22] Y.V.Matijasevic, Enumerable Sets are Diophantine, *Dokl. Akad. Nauk SSSR*, Vol.191, 279-282(1970), in Russian, English Translation in *Soviet Math. Dokl.*, Vol. 11, 354-357.

[23] R.McNaughton, Scheduling with Deadlines and Loss Functions, *Management Sci.*, Vol.6, 1-12(1959).

[24] N.Megiddo, Combinatorial Optimization with Rational Objective Functions, *Math. Operations Res.*, Vol.4, 414-424(1979).

[25] S.Sahni and Y.Cho, Scheduling Independent Tasks with Due Times on a Uniform Processor System, *J. Assoc. Comput. Mach.*, Vol.27, 550-563(1980).

[26] A.Turing, On Computable Numbers, with an Application to the Entscheidungs Problem, *Proc. of the London Math. Soc. Ser. 2*, Vol.42, 230-265(1936).

LIST OF PUBLICATIONS

[1] Some Bounds on Approximation Algorithms for $n/m/I/L_{max}$ and $n/2/I/L_{max}$ Scheduling Problems, *Journal of the Operations Research Society of Japan*, Vol.26, 212-224(1983).

[2] Two-Machine Scheduling Problem for Jobs with Generalized Due Dates, *Mathematica Japonica*, Vol.30, 117-125(1985).

[3] A Generalized Uniform Processor System, *Operations Research*, Vol.33, 346-362(1985).

[4] The Mixed Shop Scheduling Problem, *Discrete Applied Mathematics*, Vol.11, 175-186(1985).

[5] Two Machine Mixed Shop Scheduling Problem with Controllable Machine Speeds. (submitted).

[6] A Solvable Case of Three Machine Open Shop Scheduling to Minimize Maximum Completion Time. (in preparation).

[7] Scheduling to Minimize Maximum Completion Time on Quasi-Identical Parallel Machines. (in preparation).

ACKNOWREDGEMENTS

First of all, the author would like to express his sincere appreciation to Professor T. Nishida for supervising this thesis. His continuous encouragement and invaluable comments have helped to accomplish the thesis.

The author also wishes to acknowledge Professor H. Sugiyama, Professor M. Yamamoto and Professor Y. Tezuka for their useful advices and helpful comments for improving the thesis.

The author is also indebted to Associate Professor H. Ishii who guided the author to the present study and has been giving the continuous encouragement and invaluable comments to complete the thesis.

The author also wishes to thank Associate Professor Y. Tabata for his continuous encouragement and useful suggestions.

Furthermore, the author would like to express his heartfelt gratitude to Professor N. Hagiwara of Osaka Prefecture University, who gave the author innumerable advices and comments for correcting some of grammatical errors and improving the author's poor English in the first draft of the thesis.

Finally, the author wishes to thank Dr. F. Ohi, Dr. S. Shiode, and the members of Nishida's Laboratory of Osaka University for their continuous kindness and friendship.