MULTI-PROCESSOR TASK SCHEDULING WITH MAXIMUM TARDINESS CRITERIA

By

WONG Tin-Lam

Department of Systems Engineering and Engineering Management

A DISSERTATION SUBMITTED TO

THE CHINESE UNIVERSITY OF HONG KONG

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF PHILOSOPHY

June 1998

Abstract

This thesis is concerned with the multi-processor task scheduling problem in which some jobs may need the processing of multiple machines simultaneously. Specifically, we investigate the following model: A number of jobs, each with its own processing requirement and due-date, are to be processed by two uniform machines. Job preemption is allowed. There are two types of jobs, *small* and *big*. Each *small* job occupies one machine only. Some *small* jobs are dedicated to a prespecified machine while the others can be processed by any one of the two machines. Each *big* job is to be processed by both machines simultaneously. The objective is to minimize the maximum tardiness. The problem is shown to be NP-hard. A pseudo-polynomial algorithm is derived to solve the problem. We further find that the problem becomes polynomially solvable when all *small* jobs have equal processing times and when both machines are identical. We also investigate the model where only *set* jobs are involved. Each *set* job can be processed by either one machine or both machines simultaneously. We will further investigate the cases when the number of machines is increased to k and when no preemption is allowed.

Acknowledgement

I would like to express my greatest gratitude to my supervisor, Dr. X. Q. Cai, for his concern, enthusiasm and belief in this area of work which has been an enormous support to me throughout the research programme.

Thanks must be given to Prof. C.-Y. Lee of Texas A&M University for his invaluable comments and constructive suggestions which proved to be very helpful to my research. I am also very grateful to Dr. H. M. Yan and Dr. D. Li who have shown understanding and made the submission of this thesis possible.

I would also like to acknowledge my friends and colleagues, with whom I have shared enjoyment and encouragement.

Finally, I must thank my parents, sisters and Wu who have been supportive and understanding.

Contents

List of Figures

Chapter 1

Introduction

1.1 Scheduling Problems

Machine scheduling is to study the problem of optimally allocating resources available to process tasks. In terms of resources, we may mean machines in a factory, processors in a computing environment, or loading and unloading facilities in a goods transportation system, and so on. Correspondingly, tasks may be the operations required by products, the executions of computer programs, or goods in transportation systems, and so on. Scheduling is involved in nearly all kinds of practical manufacturing environment as well the service industries.

In the classical scheduling theory, the simplest model is the single machine problem in which only one machine is available to process a set of jobs. Each job is processed by the machine and there is at most one job occupying the machine at any

time. Scheduling of arrival and departure times of ferries at a pier is an example of a single machine scheduling problem. Apart from the single machine model, the problem with multiple machines processing in parallel is referred to as a parallel machine scheduling model. In a parallel machine model, jobs can be allocated to one of the available machines. An example of a parallel machine scheduling model is the scheduling of the data transmission in a computer network with multiple gateways.

Many scheduling models have been studied in the literature, see [1, 13, 22]. Nevertheless, most scheduling literature assumes the one-job-on-one-machine structure where at any time, a machine can process only one job and a job can only be processed by one machine. Such a model has a restriction that all jobs involved must be one-machine jobs.

In recent years, much attention has been paid to the *multiprocessor task scheduling* problem, or *1-job-on-r-machine* problem, where $r \neq 1$. When r is less than 1, a job may occupy less than one machine and the machine may process more than one job at any time. An example is a cargo forwarding problem, suggested by Li et al in [20], where a number of trucks carrying different kinds of cargos must be discharged. On the other hand, when r is bigger than 1, a job must occupy more than one machine at the same time, or equivalently, more than one machine is necessary to process the same job. The problem studied in this thesis is a *1-job-on-r-machine* case, where $r > 1$. This is motivated by scheduling problems in environments such as berth

allocation in which vessels are to be allocated to berths for loading and/or unloading under certain criteria(objectives). In the berth allocation problem, the lengths of the vessels are not standardized, and different vessels may have different lengths. For a vessel of shorter size, one berth is good enough, while a longer vessel must occupy more berths. Hence, this is a multi-processor task scheduling problem where jobs(vessels) may occupy one or more machines(berths) for simultaneous processing. In fact, multiprocessor task scheduling problems exist in many real life situations. Another example is the human resources planning problem with limited staff available to process up-coming projects. The problem of how to schedule the workforce to complete tasks which require one or more people for simultaneously processing falls into our one-job-on-r-machine model. Other examples include the diagnosable microprocessor systems (see [16]), where a number of machines have to work in parallel in order to find a fault, and parallel computer systems, where several processors can work simultaneously to execute a single job submitted by a user. As we can see, these applications are not amenable by the one-job-on-one-machine pattern, where a job can be processed on only one machine at any time. There is a great need to study and solve one-job-on-multiple-machine problems. This is a new direction in the area of scheduling.

1.2 Literature Review

Three classes of *multiprocessor task scheduling* problems have been studied in the literature. The first class of the problems assume that each job needs a fixed number of processors to work simultaneously, where allocations of processors are not specified. The second class of problems assume that each particular job requires certain specific machines for processing. For a example, if there are three machines available, M_1, M_2 and M_3, a certain job may need to be processed by M_2 and M_3 simultaneously, yet another job may be dedicated to M_1 only, etc. We use *dedicated* or *prespecified machine(s)* to refer to the specific machines throughout this thesis. Lastly, the third class of problems deal with the situations where the allocation of a job to the machines has several alternatives. In each alternative, a job can be processed by several machines simultaneously. We call the first, second and third classes of problems as *sized*, *fixed* and *set multiprocessor task scheduling* problems, respectively.

We follow the notation used by [22] to denote a scheduling problem by a triplet $\alpha \mid \beta \mid \gamma$. The α field, a single entry, describes the machine environment. The β field provides details of processing characteristics and constraints, which can contain more than one entry. Finally, the γ field, also a single entry, contains the objective to be minimized. The descriptions $size_j$, fix_j and set_j are used in the second field of $\alpha \mid \beta \mid \gamma$ to denote the first, second and third classes of problems respectively. The descriptions $size_j$, fix_j and set_j used in the β field to denote the three classes of problems.

In the following sections, we shall review the research on the three classes of problems.

1.2.1 Sized Multiprocessor Task Scheduling

Blazewicz, Drabowski and Weglarz [8] study the problem of minimizing the schedule length of scheduling n jobs on m machines. The jobs considered may occupy one or more arbitrary machine(s) at the same time. In the nonpreemptive case, two polynomial time algorithms are given when all jobs needs equal processing times, while the problem is shown to be NP-complete when jobs have arbitrary processing times. For the preemptive case, a polynomial time algorithm is given when jobs require one or k machines and have arbitrary processing times. Finally, the possibility of a linear programming formulation for the general case is also discussed.

Lee and Cai [18] consider the problem with two criteria, the total weighted completion time and the maximum lateness. Two types of jobs, namely one-processor jobs and two-processor jobs, are considered, . The problems are shown to be NP-hard in the strong sense for both criteria even when there are only two machines. Dynamic programming algorithms, with pseudo-polynomial time complexity, are presented while a polynomial algorithm is provided to solve a special case where all jobs have equal processing times. Heuristic methods are also presented with error bounds

established. The algorithms provided for the two-machine problem are shown to be extendable to deal with the general problem.

The problem where jobs can be executed by one or more processors at the same time with the objective to minimize schedule length is studied in [12]. Du and Leung [12] show that pseudo-polynomial time algorithm exist for 2- and 3-machine problems, while the problem is NP-hard in the strong sense where there are 5 machines and preemption is not allowed. The question whether the 4-machine problem is NP-hard in the strong sense remains open. For the preemptive case, the problem is shown to be NP-hard in the strong sense for arbitrary number of processors solvable in pseudo-polynomial time for a fixed number of processors.

Plehn [23] shows that the problem of preemptive scheduling of independent jobs with release times and deadlines on a hypercube can be formulated as a linear program. Each of the job considered, which has its own release time, requires a set of processors for processing simultaneously. The case of general release times and deadlines is shown to be solvable by a linear programming approach.

1.2.2 Fixed Multiprocessor Task Scheduling

The problem of preemptive scheduling of tasks requiring a set of processors simultaneously to minimize maximum lateness is studied in [3]. Tasks can be processed by

one or more set(s) of prespecified processors. When there is only one feasible set of prespecified processors for all jobs, the model is a *fixed multiprocessor task scheduling*. Otherwise, it is a *set multiprocessor task scheduling*. Bianco et al [3] formulate the *fixed* problem in terms of linear programming and show that it can be solved in polynomial time in the number of tasks.

Bianco, Blazewicz, Dell'Olmo and Drozdowski [5] consider the problem of scheduling tasks that require more than one dedicated processor at a time to minimize the maximum lateness. Linear time algorithms are given for the case of two, three and four processors which deliver optimal solutions in some cases but have no guarantee on optimality in other cases.

Blazewicz et al [7] study the problem of scheduling a set of multiprocessor tasks on three dedicated processors is studied. The tasks considered require simultaneous availability of a specified subset of processors and minimization of the makespan of the schedule is the objective criterion. The general problem of scheduling tasks on three dedicated processors is proved to be NP-hard in the strong sense and an approximation algorithm is given and analyzed.

Cai, Lee and Li [10] consider the problem of scheduling multiprocessor tasks with prespecified processor allocations, where the total completion time is the objective

criterion. The complexity of both preemptive and nonpreemptive cases of the two-processor problem are studied. They show that the preemptive case can be solved in polynomial time for two machines while it remains an open question of whether the problem is solvable in polynomial time when the number of machine is more than two. Moreover, they also show that the problem is NP-hard in the strong sense for nonpreemptive case.

The computational complexity of preemptive and nonpreemptive scheduling of biprocessor tasks on dedicated processors is studied in [17]. Kubale [17] consider two criteria of optimality: the makespan and the total completion time. Each task considered requires the simultaneous execution of two prespecified processors, while each processor can execute at most one task at a time. They also analyze the complexity of these problems when precedence constraints are involved. In addiction, they show that in general all these problems are NP-hard in the strong sense.

1.2.3 Set Multiprocessor Task Scheduling

The *set* problem of scheduling preemptable tasks requiring a set of processors simultaneously so as to minimize maximum lateness is studied in [3]. Tasks can be processed by one of many feasible sets of prespecified processors in this *set multiprocessor task scheduling* model. Bianco et al [3] formulate the *set* problem in terms of linear programming and show that it can be solved in polynomial time in the number of tasks.

A nonpreemptive scheduling problem in which a set of independent tasks must be processed on a set of discrete and renewable resources is studied in [6]. Each resource concerned can be used at any time by a single task at most while each task can be carried out in several given alternative modes, that is, by using different resource sets and with different processing times. The objective is to determine a mode and a starting time for each task in such a way that the makespan is minimized. The authors study the complexity of the problem and several cases are identified as NP-hard.

In [11], Chen and Lee consider the problem where there are several alternatives that can be used to process each job. In each alternative, several machines need to process simultaneously the job assigned. The objective is to select an alternative for each job and then to schedule jobs to minimize the maximum completion time of all jobs. A pseudo-polynomial algorithm is provided to solve optimally the two-machine problem, and a combination of a fully polynomial scheme and a heuristic to solve the three-machine problem is also proposed. The results are then extended to a general $m-$machine problem.

A comprehensive review on problems of scheduling multiprocessor tasks is given by [9]. A summary of the current trends of multiprocessor tasks scheduling and machine scheduling with availability constraints is given in [19].

The problem we are going to deal with in this thesis concerns with scheduling of two types of preemptive jobs on two uniform machines, so as to minimize the maximum tardiness. Based on our review of the scheduling literature, this problem is a new scheduling model, which however represents an important category of practical applications. In the following chapters, we will report our work in problem modeling, solution methods development and extensions.

1.3 Organization of Thesis

The remainder of the thesis is organized as follows: Chapter 2 presents the model. Chapter 3 is methodology development. In Chapter 4 we consider special well-solvable cases and extensions of our model and solution method. Finally, some concluding remarks is given in Chapter 5.

Chapter 2

Overview

In this chapter, we formulate our problem, which includes the machine environment, the jobs and their requirements, the assumptions, the constraints, the objective function, and a mathematical model with terminologies and variables. Some remarks are also given in the final section.

2.1 Machine Environment

Two parallel machines, which may have different processing speeds, are to process a group of jobs. Since their processing speeds may be different, they are referred to as uniform machines. The processing speed of machine i is denoted by v_i, where $i = 1, 2$. Note that when $v_1 = v_2$, the problem reduces to one with parallel identical machines.

In the notation $\alpha \mid \beta \mid \gamma$, the field α now is denoted as Qm, where $m = 2$. This indicates that the problem under consideration has two uniform machines.

2.2 The Jobs and Their Requirements

Jobs are grouped into one-machine jobs and two-machine jobs and are referred to as *small* jobs and *big* jobs throughout this thesis. There are a total of n jobs including *small* and *big* jobs. There are two types of *small* jobs, dedicated and non-dedicated. At any time, each dedicated *small* job has to be processed by a machine that is prespecified to it. A non-dedicated job can be processed by either one of the two parallel machines. On the other hand, *big* jobs have to be processed by both machines simultaneously. This means that they will start, complete and preempt, if applicable, simultaneously among the two machines.

Each job j has a fixed amount of processing requirement and a due date, which are denoted by p_j and d_j respectively. We let p_{ij} denote the processing time of *small* job j if it is processed on machine i, which is equal to p_j/v_i. All *small* jobs must be processed by the assigned machine throughout its operation. When all machines are identical, v_i is constant for all i and hence we may scalize the speed to unity and let the processing time be p_j. In the case that *big* jobs are processed by both machines, it is assumed that the time each *big* job j spends on both machines is denoted by $p_j^b = p_j$.

As one can see, our problem does not belong to *fixed* or *sized*. The *fixed* problem has jobs which are dedicated to a set of prespecified machine(s), but in our problem there are jobs that can be assigned to either machines. The *sized* problem has jobs which can be processed by a fixed number of arbitrary machines, but in our problem there are jobs which must be processed by a pre-specified machine. Therefore, we denote our problem as a *mixed multiprocessor tasks scheduling* problem and use mix_j in the second field of the notation $\alpha \mid \beta \mid \gamma$ to represent this class of scheduling problems.

2.3 Assumptions

The following are some assumptions apply to the main problem as well as other extensions studied in this thesis.

1. All jobs are simultaneously available at the beginning, i.e. $r_j = 0$ for all $j = 1, 2, ..., n$. This assumption is valid when all jobs arrive before $t = 0$ and are ready for processing at $t = 0$.

2. All processing requirements, p_j, of the jobs are known as we are dealing with a deterministic problem.

3. All jobs have equal weights and are assumed to be unity. The assumption is justified when the importance of all job in the system are about the same.

4. Job preemption is allowed, which means that the processing of jobs may be interrupted before it is complete. We further that the cost incurred from preemption is negligible. In the β field of the $\alpha \mid \beta \mid \gamma$ notation, we will use *prmp* to represent that preemption is allowed.

5. No precedence relationships exit among jobs. Jobs considered in our problem, are assumed to be independent in the sense that no job or jobs are required to be completed before another job is allowed to start its processing.

6. The processing speed for each machine is known in advance.

7. The set-up time to process a job has been included in its processing time.

8. No machine breakdowns occur, which imply that machines are continuously available.

2.4 Constraints

There are some constraints in our problem which are listed below.

1. A *big* job must occupy both machines when being processed. In general, in the

$k-$machine case where *big* jobs refer to $k-$processor jobs, a *big* job must occupy all k machines simultaneously.

2. A *small* job must be processed by the assigned machine throughout the operation even though it is preempted. The validity of this assumption is well justified in the situation where there is a large switching cost (incurred probably by the need of transportation, set-up, etc.) if a job is switched from one machine to another machine. Consider the environment of berth allocation with two berths, where the two berths (machine 1 and machine 2) normally adjoin each other. When the processing by machine 1 (or 2) of a *small* vessel is preempted, it should move towards the opposite direction of machine 2 (or 1) in order not to interrupt any on-going or up-coming operations of both machines. Then, if it is brought back into operation on machine 2 (or 1), it has to pass through the vessels occupying machine 1 (or 2) and this is rather interruptive to operations in all two berths and is not a desirable situation.

2.5 Objective

The objective is to schedule jobs on two uniform parallel machines such that the maximum tardiness of all jobs is minimized. By saying that a job is tardy, we mean that it completes after its due date. The tardiness of a job, T, is defined as the

lateness of the job, L, if it fails to meet its due date or zero otherwise:

$$T = \max\{0, L\}, \tag{2.1}$$

where the lateness of a job, L, when its due date is d and completion time is c, is:

$$L = c - d. \tag{2.2}$$

Note that the lateness of a job can be negative, while its tardiness is always non-negative.

The maximum tardiness, $T_{\max}$, under a schedule involving n jobs is defined as:

$$T_{\max} = \max\{T_1, T_2, ..., T_j, ..., T_n\}, \tag{2.3}$$

where T_j is the tardiness of job j for $j = 1, 2, ..., n$.

Since $T_{\max}$ is our objective function, in the γ field of the $\alpha \mid \beta \mid \gamma$ notation is $T_{\max}$. In summary, the problem we are investigating is $Q2 \mid mix_j, prmp \mid T_{\max}$.

2.6 An Illustrative Example

In this section, we will give an example to illustrate the model $Q2 \mid mix_j, prmp \mid T_{\max}$ we described above.

Two machines, M_1 and M_2, are to work in parallel to process a group of n jobs containing *small* and *big* jobs. The processing speeds of M_1 and M_2, v_1 and v_2 are respectively:

$$\begin{aligned} v_1 &= 1, \\ v_2 &= 2. \end{aligned}$$

There are three *small* jobs and two *big* jobs where job j is denoted by J_j, and J_1, J_2 and J_3 are *small* jobs while J_4 and J_5 are *big* jobs. The processing requirements, p_j, of the jobs are as follows:

$$\begin{aligned} p_1 &= 4; \\ p_2 &= 6; \\ p_3 &= 1; \\ p_4 &= 6; \\ p_5 &= 12. \end{aligned}$$

And the corresponding due dates, d_j, of the jobs are:

$$d_1 = 3;$$

$$d_2 = 8;$$

$$d_3 = 2;$$

$$d_4 = 4;$$

$$d_5 = 5;$$

Small jobs J_1 and J_2 are non-dedicated *small* jobs while J_3 is dedicated to Machine 1. The objective is to minimize the maximum tardiness:

$$T_{\max} = \max\{T_1, T_2, T_3, T_4, T_5\},$$

where

$$T_j = \max\{0, c_j - d_j\} \qquad \text{for } j = 1, 2, ..., 5$$

in which c_j and d_j are the completion time and due date of job j.

As mentioned before, *small* jobs J_1 and J_2 can be processed by machine 1 or machine 2. If J_1 is processed by machine 1, it will need a processing time $p_1/v_1 = 4/1 = 4$ to complete. Instead, if it is processed by machine 2, it will need a processing time $p_1/v_2 = 4/2 = 2$. Similarly, one can obtain the processing times of J_2 on machines 1 and 2. *Small* job J_3 which is dedicated to machine 1 must occupy machine 1 and

need a processing time $p_3/v_1 = 1/1 = 1$. Finally, *big* jobs J_4 and J_5 will occupy both machines simultaneously and require processing times $p_4^b = p_4 = 6$ and $p_5^b = p_5 = 12$, respectively.

By the assumptions, all the jobs are available at the beginning. All *small* job J_1, J_2 and J_3 must be finished by the assigned machine throughout operation. Moreover, *big* jobs J_4 and J_5 must occupy both machines simultaneously. One of the feasible schedule:

M_1 : $J_1(t = 0 \text{ to } 4)$, $J_3(t = 4 \text{ to } 5)$, $J_4(t = 5 \text{ to } 11)$, $J_5(t = 11 \text{ to } 23)$.

M_2 : $J_2(t = 0 \text{ to } 3)$, (idle for $t = 3$ to 5), $J_4(t = 5 \text{ to } 11)$, $J_5(t = 11 \text{ to } 23)$.

In the above schedule, J_1 starts at $t = 0$ on machine 1 and completes at $t = 4$. J_2 starts at $t = 0$ on machine 2 and completes at $t = 3$. J_3 starts at $t = 4$ on the prespecified machine, M_1, and finishes at $t = 5$. *Big* job J_4 starts on both machines at $t = 5$ and completes at $t = 11$, *big* job J_5 starts on both machines at $t = 11$ and finishes at $t = 23$. Note that M_2, machine 2, is idle during the period $t = 3$ to 5. The tardiness, T_j, of the jobs can be obtained as follows:

$$\begin{aligned} T_1 &= \max\{0, c_1 - d_1\} = \max\{0, 4 - 3\} = 1; \\ T_2 &= \max\{0, c_2 - d_2\} = \max\{0, 3 - 8\} = 0; \\ T_3 &= \max\{0, c_3 - d_3\} = \max\{0, 5 - 2\} = 3; \end{aligned}$$

$$T_4 = \max\{0, c_4 - d_4\} = \max\{0, 11 - 4\} = 7;$$

$$T_5 = \max\{0, c_5 - d_5\} = \max\{0, 23 - 5\} = 18.$$

Thus the maximum tardiness among all jobs is:

$$\begin{aligned} T_{\max} &= \max\{T_1, T_2, T_3, T_4, T_5\} \\ &= \max\{1, 0, 3, 7, 18\} \\ &= 18. \end{aligned}$$

This means that the schedule, which satisfies all constraints, has a maximum tardiness value equal to 18.

2.7 NP-Hardness

In this section, we discuss the complexity of our problem as to NP-hardness. For a review on NP-hardness, please see [14].

We can show that our problem is an NP-hard problem. Note that if we consider a problem with only non-dedicated *small* jobs involved, then the problem is actually a $Q2 \mid\mid T_{\max}$ problem. Since later we will show that the optimal schedule should not have any idle time or preemption inserted for $Q2 \mid\mid T_{\max}$, the problem equivalent to a non-preemptive case. The $P2 \mid\mid T_{\max}$ problem is equivalent to $P2 \mid\mid L_{\max}$ except the slight difference in the objective measure. As we mentioned earlier, the definition of

the lateness of a particular job j, L_j, is given by:

$$L_j = c_j - d_j, \tag{2.4}$$

where c_i and d_i are the completion time and due date of job i, respectively. The maximum tardiness of a schedule, $L_{\max}$, is given by:

$$L_{\max} = \max\{L_1, L_2, ..., L_j, ..., L_n\} \qquad \text{for } j = 1, 2, ..., n, \tag{2.5}$$

where n is the total number of jobs.

The $P2 \mid\mid T_{\max}$ problem, and the $Q2 \mid\mid T_{\max}$, are not simpler than $P2 \mid\mid L_{\max}$. Furthermore, the problem we are investigating is not simpler than $Q2 \mid\mid T_{\max}$. Since $P2 \mid\mid L_{\max}$ is an NP-hard problem (see [22]), the problem under study is also NP-hard.

Chapter 3

Methodology

In this chapter, we will first briefly review the technique of Dynamic Programming, the major tool we use to develop our algorithm. We will then analyze our problem and discuss the key idea to solve our problem. Finally, the last section is a detail description of our algorithm.

3.1 Dynamic Programming

Dynamic Programming (DP) is a useful mathematical technique for dealing with multi-stage decision making problems. It provides a systematic procedure for determining the optimal combination of decisions. It was proposed by R. Bellman in the 50's, (see [2]). The two fundamental principles of DP are the *principle of decomposability* and the *principle of optimality.* The first principle requires that the

problem be first decomposable into stages. Then the second principle guarantees an optimal solution by applying an optimal policy at each stage. An optimal policy has the property that whatever the initial stage and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

The basic features that characterize Dynamic programming consist of:

1. The problem is divided into stages and states with a policy decision required at each stage for different states.

2. A recursive relationship is constructed which identifies the optimal policy for stage i, if that for stage $i-1$ is available.

3. The optimal value of the objective function is obtained by applying the recursive relationship with initial conditions.

4. The optimal solution is obtained by backtracking.

In this thesis, we use DP approach to solve the $Qm \mid\mid T_{\max}$ problem. The problem is first divided into an $n-$stage of decision problem. The decision of which machine the ith job should be assigned to is determined at the ith stage $i = 1, 2, ..., n$. At each stage there are different possible states represented by a set of state variables. Each state variable corresponds to the completion time associated with one of the m machines, while the m completion times are contributed by the total processing time of the first i jobs at the ith stage. At each stage, the optimal decision for all possible

states are evaluated by applying a recursive relationship.

Finally, the optimal solution can be obtained by a backtracking procedure. There is an optimal value associated with each possible state at the final stage, the nth stage. The optimal value is given by the minimum among all possible states. The optimal decision for the nth job and optimal state for the $n-1$th stage can be obtained by tracing the expression which represents that decision in the recursive relationship at the nth stage. The optimal decision for the $n-1$th job can be obtained similarly given the optimal state for the $n-1$th stage. The process continues until a complete solution is identified.

3.1.1 Problem Analysis

We now analyze our problem. There are three types of jobs we are dealing with, namely, the non-dedicated *small* jobs, the dedicated *small* jobs, and the *big* jobs. The non-dedicated *small* jobs have to be allocated to the available machines while the dedicated *small* jobs and the *big* jobs just occupy respectively the prespecified machines and both two machines simultaneously. Furthermore, for each machine, the jobs have to be sequenced in a way that the objective value is minimized.

A question is whether we can do the allocation and sequencing separately. Let us

first look at our objective function:

$$\min T_{\max} = \max\{T^1_{\max}, T^2_{\max}\}, \tag{3.1}$$

where $T^i_{\max}$ is the maximum tardiness of jobs assigned to machine i. For a given allocation of jobs, if we can minimize both $T^1_{\max}$ and $T^2_{\max}$, the maximum tardiness of the schedule is minimized. But how can we minimize both $T^1_{\max}$ and $T^2_{\max}$. Since each *big* job must occupy the two machine simultaneously, it may be the case that a *big* job obeys the *EDD* (earliest due-date) rule on one machine but not on the other. An *EDD* sequence is optimal to minimize the maximum tardiness as long as the allocation of jobs in the machine is given, see [22]. We cannot guarantee that each *big* job occupies both machines simultaneously and at the same time, it obeys the *EDD* rule on both machines. Apparently, the existence of the *big* jobs makes it infeasible to separate the optimal sequencing on machines from the allocation to the machines. A key idea in our approach to deal with tis problem is to split the *big* jobs into one-machine jobs, and then restore them into a feasible one after allocation and sequencing have all been completed. This idea will be elaborated later.

One characteristic of our problem $Q2 \mid mix_j, prmp \mid T_{\max}$, which has been mentioned in section 2.7 of the last chapter, should be noted, which is the equivalence of our problem to the $Q2 \mid\mid T_{\max}$ problem when it contains only *small* jobs. First of all, we know that when no *big* jobs are involved in our problem $Q2 \mid mix_j, prmp \mid T_{\max}$, it reduces to the problem $Q2 \mid prmp \mid T_{\max}$. The following Lemma then shows our

problem further reduces to $Q2 \mid\mid T_{\max}$.

Lemma 1 *No idle time or preemption is necessary for sequencing jobs on one machine to minimize maximum tardiness.*

Proof. Consider when an idle time of duration, δ, is introduced in a sequence of jobs an one machine. All jobs following the idle time have their completion times involving δ. It is easy to see that their tardiness will be decreased by an amount [0, δ] if the time is removed. This means that inserting the idle time is unnecessary as the objective value will not be increased without the idle time. On the other hand, consider a schedule with a job J_j being preempted at time t and resumed processing on the same machine after a time δ. Denote the time when J_j starts processing as s_j and the time when it completes processing as c_j. If J_j delays its start time by δ and the jobs occupying the time duration from $[t,\ t+\delta)$ are shifted to $[s_j,\ s_j+\delta)$, then J_j will occupy the machine in $[s_j+\delta,\ c_j)$ without preemption. By doing so, the completion time and thus the tardiness, of J_j is unchanged, while the completion times of the jobs previously occupying the machine at $[t,\ t+\delta)$ are decreased and thus their tardiness is not increased. This means job preemption is also unnecessary. Thus the Lemma is proved.□

3.2 Key Idea to solve the problem

In this section, we will discuss our main idea to develop an algorithm to solve our problem. The details of the algorithm and the proof of the theorems will be given in the next section.

We solve our problem in two phases. In the first phase, we develop a dynamic programming algorithm to solve the $Qm \mid\mid T_{\max}$ problem. Then in the second phase, we formulate a problem, which is a $Q2 \mid\mid T_{\max}$ problem, by relaxing some constraints of our problem $Q2 \mid mix_j, prmp \mid T_{\max}$. The optimal value of the $Q2 \mid\mid T_{\max}$ problem is proved to be a lower bound of our $Q2 \mid mix_j, prmp \mid T_{\max}$ problem. We then apply the algorithm developed in Phase 1 to solve the $Q2 \mid\mid T_{\max}$ problem. After that, the solution for $Q2 \mid\mid T_{\max}$ is adjusted to make it become a feasible solution for $Q2 \mid mix_j, prmp \mid T_{\max}$.

In our approach, the key idea is a split of each *big* job into a pair of dedicated jobs, each to one machine. This makes the multiprocessor task scheduling problem $Q2 \mid mix_j, prmp \mid T_{\max}$ into a traditional one-processor task scheduling problem $Q2 \mid\mid T_{\max}$, which is solvable by dynamic programming. After the solution to $Q2 \mid\mid T_{\max}$ is obtained, we then restore each pair of the split dedicated jobs into a *big* two-processor job. As we can prove that the restoring operation does not increase the maximum tardiness, the solution after the restoring operation becomes feasible to $Q2 \mid mix_j, prmp \mid T_{\max}$, and is optimal.

3.3 Algorithm

In this section, we present our algorithm. As discussed in the last section, we tackle the problem $Q2 \mid mix_j, prmp \mid T_{\max}$ by a two-phase approach..

3.3.1 Phase 1

It is well known that (see [13]) the maximum tardiness of jobs processed by one machine is minimized by the Earliest Due Date (EDD) Rule:

$$d_{[1]} \leq d_{[2]} \leq \cdots d_{[n]},$$

where $d_{[j]}$ is the due date of the jth job in the sequence of n jobs. In this case, the problem is easy to solve, as an EDD sequence can be constructed in $O(n \log n)$ time. However, if the problem involves more than one machine, its difficulty is totally different, wince we will have to deal with not only the sequencing of jobs on a machine, but also the allocation of jobs to the different machines. The dynamic programming algorithm we develop will deal with the allocation part. Each job will be allocated to one of the available machines and each machine will have a set of allocated jobs. After an allocation of jobs to the machine is obtained, all we have to do is to schedule the jobs allocated to each machine in EDD order such that the maximum tardiness,

$T^i_{\max}$ is minimized.

Although the main purpose of this thesis is to solve a two-machine problem, the first phase of our approach can be applied to any $m-$machine problem. In the following we will describe an algorithm for solving the $m-$machine problem first.

Dynamic Programming

We first re-index the jobs in Earliest Due Date (EDD) order, namely, the jobs are re-arranged such that:

$$d_1 \leq d_2 \leq ... \leq d_n,$$

where d_j is the due date of the jth job. Our dynamic programming algorithm can be described as follows.

Stage : We first divide the problem into *stages* $1, 2, ..., n$. Stage j corresponds to the decision of which machine job j should be assigned to. At stage 1, the allocation of job 1 is considered, and at stage 2, job 2. This continues until all the n jobs are considered.

State : At each stage, there are a number of possible conditions which are represented by *state* variables. In our problem, at every stage there are m latest completion times, each corresponding to one of the m machines. We denote the completion time on machine i by t_i.

Recursive Relationship : Define $f_j(t_1, t_2, ..., t_m)$ as the minimum $T_{\max}$ given that we have assigned jobs $1, 2, ..., j$ to the machines, and the total processing time on machine i (equal to the latest completion time on machine i) is t_i for $i = 1, 2, ..., m$. Then we can see that:

$$f_j(t_1, t_2, ..., t_m) = \min_{1 \le i \le m} \{\max\{f_{j-1}(t_1, t_2, ..., t_i - p_j/v_i, ..., t_m), \max\{0, t_i - d_j\}\}\}. \quad (3.2)$$

The above equation is obtained based on the following argument. At stage j, we have a set of latest completion times on of the m machines, $t_1, t_2, ..., t_m$. If job j is processed by machine i, then the tardiness for job j should be:

$$\max\{0, t_i - d_j\}.$$

And the processing time of job j if it is processed by machine i is given by:

$$p_{ij} = \frac{p_j}{v_i}, \quad (3.3)$$

and thus the latest completion time on machine i at the previous stage, i.e. after job $j - 1$ is assigned, is:

$$t_i - \frac{p_j}{v_i}. \quad (3.4)$$

The recursive relationship should give an optimal return at stage j given the set

of values $t_1, t_2, ..., t_m$. Since the tardiness of job j is $\max\{0, t_i - d_j\}$, and the maximum tardiness of other jobs is $f_{j-1}(t_1, t_2, ..., t_i - p_j/v_i, ..., t_m)$, we obtain the recursive relationship equation 3.2.

The above recursive relationship applies to all non-dedicated jobs. If job j is dedicated to machine i, no comparison between allocations to different machines is needed as job j must occupy machine i. In this case, the recursive relation becomes:

$$f_j(t_1, t_2, ..., t_m) = \max\{f_{j-1}(t_1, t_2, ..., t_i - p_j/v_i, ..., t_m), \max\{0, t_i - d_j\}\}. \quad (3.5)$$

Ranges of Values for t_i's : Each t_i corresponds to the latest completion time of machine i. When all machines are identical (in this case, we can assume the speed $v_i = 1, \forall i$), job j will need to be processed on a machine with a processing time equal to the processing requirement p_j. At stage j where jobs $1, 2, ..., j$ have been assigned to the m machines, the total processing requirement, P_j, of jobs 1, 2, ..., j, is given by:

$$P_j = \sum_{k=1}^{j} p_k. \quad (3.6)$$

Let Π_{ij} denote the set of jobs processed on machine i at stage j. At stage j, the total processing time for the jobs in Π_{ij} is equal to the latest completion time associated with machine i, t_i:

$$\begin{aligned} t_i &= \sum_{J_k \in \Pi_{ij}} p_{ik} \\ &= \sum_{J_k \in \Pi_{ij}} p_k \end{aligned} \quad (3.7)$$

Thus at each stage j, we have:

$$\begin{aligned} t_1 + t_2 + ... + t_m &= \sum_{J_k \in \Pi_{1j}} p_k + \sum_{J_k \in \Pi_{2j}} p_k + ... + \sum_{J_k \in \Pi_{3j}} p_k \\ &= P_j. \end{aligned} \tag{3.8}$$

Now consider the case when the machines have different processing speeds. The total processing requirement at stage j, P_j is again given by:

$$P_j = \sum_{k=1}^{j} p_k. \tag{3.9}$$

Each job j processed on machine i now has a processing time $p_{ij} = p_j/v_i$. Again, let Π_{ij} denote the set of jobs processed on machine i at stage j. The total processing time of the jobs in Π_{ij} is given by:

$$\begin{aligned} \sum_{J_k \in \Pi_{ij}} p_{ik} &= \sum_{J_k \in \Pi_{ij}} \frac{p_k}{v_i} \\ &= \frac{\sum_{J_k \in \Pi_{ij}} p_k}{v_i}, \end{aligned} \tag{3.10}$$

Since the total processing time of the jobs in Π_{ij} is equal to the latest completion time on machine i, t_i, we can determine the relationship between the total processing requirement of the jobs in Π_{ij} and t_i, as follows:

$$t_i = \sum_{J_k \in \Pi_{ij}} p_{ik}$$

$$= \frac{\sum_{J_k \in \Pi_{ij}} p_k}{v_i}. \tag{3.11}$$

Or:

$$t_i \times v_i = \sum_{J_k \in \Pi_{ij}} p_k \tag{3.12}$$

Thus the total processing requirement at stage j, P_j, is given by:

$$\begin{aligned} P_j &= \sum_{J_k \in \Pi_{1j}} p_k + \sum_{J_k \in \Pi_{2j}} p_k + ... + \sum_{J_k \in \Pi_{mj}} p_k \\ &= t_1 \times v_1 + t_2 \times v_2 + ... + t_m \times v_m \end{aligned} \tag{3.13}$$

In particular, the value of t_i can be written as:

$$t_i = \frac{P_j - \sum_{k=1, k \neq i}^{m} t_k \times v_k}{v_i}. \tag{3.14}$$

We know from equation (3.13) that there is an upper bound for the values of the latest completion times t_i's for each stage j. First, it is clear that:

$$t_i \geq 0, \qquad \forall i = 1, 2, ..., m. \tag{3.15}$$

For machine i, the maximum value of t_i is bounded above by equation (3.14) such

that:

$$t_i \leq \frac{P_j - \sum_{k=1}^{i-1} t_k \times v_k}{v_i}. \tag{3.16}$$

Thus at each stage j, we have to consider the following possible values for t_is:

$$t_i = 0, 1, 2, ..., (P_j - \sum_{k=1}^{i-1} t_k \times v_k)/v_i. \qquad \forall i = 1, 2, ..., m-1 \tag{3.17}$$

Given t_i, $i = 1, 2, ..., m-1$, t_m is fixed by the following:

$$t_m = \frac{P_j - \sum_{i=1}^{m-1} t_i \times v_i}{v_m}. \tag{3.18}$$

Initial Conditions : Clearly we should have:

$$f_0(t_1, t_2, ..., t_i, ..., t_m) = 0 \qquad \forall t_i = 0,\ i = 1, 2, ..., m. \tag{3.19}$$

And

$$f_0(t_1, t_2, ..., t_i, ..., t_m) = \infty \qquad \forall t_i \neq 0,\ i = 1, 2, ..., m. \tag{3.20}$$

Optimal Value : The optimal value, $T_{\max}$, is given by:

$$T_{\max} = \min\{f_n(t_1, t_2, ..., t_m)\}. \tag{3.21}$$

Optimal Solution - Back Tracking : To obtain the optimal schedule, we have to do a *back tracking.* The procedure starts from the final stage. At each stage the optimal decision for that stage is identified as the decision that gives the optimal value at that stage.

The whole algorithm can be now described as follows:

Algorithm 1

Step 1: Re-index all jobs in Earliest Due Date (EDD) order.

Step 2: Set $f_0(t_1, t_2, ..., t_i, ..., t_m)$ *according to the initial condition given.*

Step 3: For $j = 1$ *to* n,

$t_1 = 0$ *to* P_j/v_1,

$t_2 = 0$ *to* $(P_j - t_1 \times v_1)/v_i$,

$\vdots$

$t_{m-1} = 0$ *to* $(P_j - \sum_{i=1}^{m-2} t_i \times v_i)/v_{m-1}$,

Set $t_m = (P_j - \sum_{i=1}^{m-1} t_i \times v_i)/v_m$, *and*

if job j *non-dedicated, compute:*

$$f_j(t_1, t_2, ..., t_m) = \min_{1 \le i \le m} \{\max\{f_{j-1}(t_1, t_2, ..., t_i - p_j/v_i, ..., t_m), \max\{0, t_i - d_j\}\}\}.$$

else compute: (job j is dedicated to machine i)

$$f_j(t_1, t_2, ..., t_m) = \max\{f_{j-1}(t_1, t_2, ..., t_i - p_j/v_i, ..., t_m), \max\{0, t_i - d_j\}\}.$$

Step 4: Set the optimal $T_{\max}$ as the minimum of all f_n over all t_i:

$$T_{\max} = \min_{t_1, t_2, ..., t_m} \{f_n(t_1, t_2, ..., t_m)\}.$$

Step 5: Obtain the state variables, $t_1^n, t_2^n, ..., t_m^n$, for stage n, where:

$$\begin{aligned} T_{\max} &= \min_{t_1, t_2, ..., t_m} \{f_n(t_1, t_2, ..., t_m)\} \\ &= f_n(t_1^n, t_2^n, ..., t_m^n) \end{aligned}$$

Step 6: For $j = n$ to 1, do the following: (Generate the optimal schedule by back-tracking.)

If job j is non-dedicated, assign job j to machine i if:

$$f_j(t_1^j, t_2^j, ..., t_m^j) = \max\{f_{j-1}(t_1^j, t_2^j, ..., t_i^j - p_j/v_i, ..., t_m^j), \max\{0, t_i^j - d_j\}\}$$

else assign job j to the prespecified machine.

Obtain the state variables, $t_1^{j-1}, t_2^{j-1}, ..., t_m^{j-1}$, for stage j:

$$f_j(t_1^j, t_2^j, ..., t_m^j) = \max\{f_{j-1}(t_1^{j-1}, t_2^{j-1}, ..., t_m^{j-1}), \max\{0, t_i - d_j\}\},$$

where job j is assigned to machine i.□

The Complexity of the Algorithm

There are m variables t_i at each stage, and each t_i is bounded by P_j/v_i for $i = 1, 2, ..., m-1$, which is the case when all jobs are processed by machine i. The value of t_m can be obtained from equation (??). If there is a total of n jobs, the optimal value is the minimum obtained in the nth stage, namely,

$$\text{Optimal value } = \min_{t_1, t_2, ... t_m} \{f_n(t_1, t_2, ..., t_m). \tag{3.22}$$

Thus the algorithm has complexity $O(nP^{m-1})$, where P is the total processing requirement of the n jobs.

3.3.2 Phase 2

Formulation of a New One-processor Problem

Consider a problem, which is identical to our problem $Q2 \mid mix_j, prmp \mid T_{\max}$ except that the *big* jobs are not necessary be processed simultaneously on both machines. We call this Problem 1. Although preemption is allowed in Problem 1, the constraint that all *small* jobs must be processed by one machine still applies. Clearly, the optimal value of Problem 1 is a lower bound of that of our problem $Q2 \mid mix_j, prmp \mid T_{\max}$ since a constraint in our problem is relaxed in Problem 1.

Problem 1 is equivalent to $Q2 \mid\mid T_{\max}$ problem

In Problem 1, we can consider each *big* job as having two "parts", one is processed by machine 1 and the other by machine 2. The two parts must spend the same time on each machine, which should be equal to that of the original *big* job. Note that in Problem 1, the two parts need not occupy both machines simultaneously. We can treat each of the *big* job in Problem 1 as two separate dedicated *small* jobs. This is achieved by spliting each *big* job into a pair of dedicated *small* jobs, with one being dedicated to machine 1 and the other to machine 2. Each split job has the same processing time and due date as the original *big* job. Note that for each pair dedicated *small* jobs, the processing requirements have to be computed such that when they are put on the prespecified machine, the processing time required will be equal to that of the original *big* job. The following gives an example on how this is computed.

Example : Let *big* job j has processing requirement $p_j = 6$ and a due date $d_j = 4$. Suppose the processing speeds of the two machines, v_1 and v_2, are 1 and 2 respectively, and the speed of the two machines, when a *big* job is processed simultaneously, is $v_3 = 3$. The processing time of *big* job j, p_j^b should be:

$$\begin{aligned} p_j^b &= \frac{p_j}{v_3} \\ &= \frac{6}{3} \\ &= 2 \end{aligned}$$

When *big* job j is split into two dedicated *small* jobs J_k and J_l, with J_k dedicated

to machine 1 and J_l dedicated to machine 2, both *small* jobs should have due dates $d_k = d_l = 4$ and processing times $p_{1k} = p_{2l} = 2$. Then the processing requirements for jobs J_k and J_l, denoted as p_k and p_l, are:

$$\begin{aligned} p_{1k} &= \frac{p_k}{v_1} \\ &\Rightarrow\ 2 = \frac{p_k}{1} \\ &\Rightarrow\ p_k = 2, \end{aligned}$$

and

$$\begin{aligned} p_{2l} &= \frac{pl}{v_2} \\ &\Rightarrow\ 2 = \frac{p_l}{3} \\ &\Rightarrow\ p_l = 6. \end{aligned}$$

Now, Problem 1 consists of a set of dedicated and non-dedicated *small* jobs since all *big* jobs are split into dedicated *small* jobs. As indicated in section 3.1.1, Problem 1 is actually equivalent to the $Q2 \mid\mid T_{\max}$ problem. Hence, its solution can be obtained by applying the algorithm developed in Phase 1.

Adjustments to *Big* Jobs

In an optimal schedule of Problem 1 obtained by algorithm 1, each pair of the dedicated jobs corresponding a same *big* job may not start and finish at the same times. We now introduce an adjustment, to move such a pair of jobs to occupy the

same time slot on both machines. Consider a *big* job, split into two *small* jobs with one dedicated to machine 1 and the other to machine 2 respectively. If the two split jobs are not scheduled to start and complete at the same times, see figure 1, then there is one, say, job a, which starts and completes earlier than the other, job b. Adjust job a and the sequence of jobs following job a to start at the starting time of job b, see figure 2. There will be a gap between the old and the new starting time of job a. Let the length of the gap be θ. The gap θ can be filled by a subsequence of jobs following job a, with length θ, and the remaining subsequence will shift to start earlier right after job a, see figure 3. Note that the remaining subsequence of jobs are unchanged before and after the adjustment described.

Algorithm 2

The following is the algorithm for solving our problem $Q2 \mid mix_j, prmp \mid T_{\max}$.

Step 1: Split each big job into a pair of dedicated small jobs, one dedicated to machine 1 and the other to machine 2. Both have the same processing times and due dates as the original big job.

Step 2: Call Algorithm 1, to obtain the optimal schedule to the new problem with only one-processor jobs (Problem 1).

Step 3: Adjust each pair of dedicated jobs corresponding to a same big jobs to start and complete at the same time as described above such that all big jobs occupy both machines at the same time.□

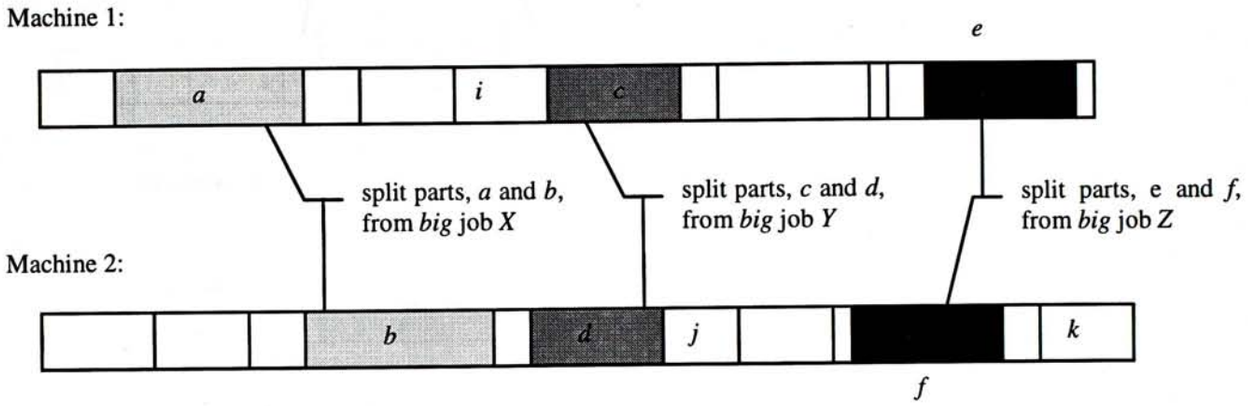

Figure 3.1: Optimal schedule for the original set of small jobs and the set of split dedicated *small* jobs

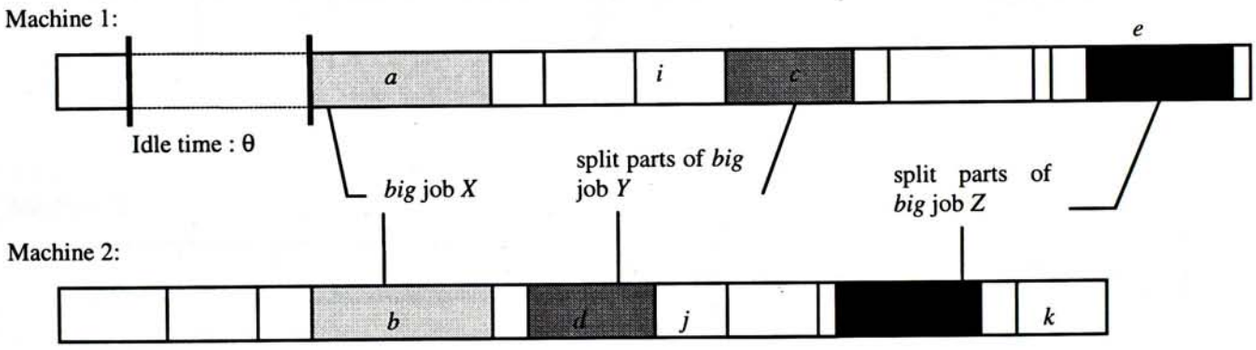

Figure 3.2: Adjust job i and the sequence of jobs following job a to start at the time job b does

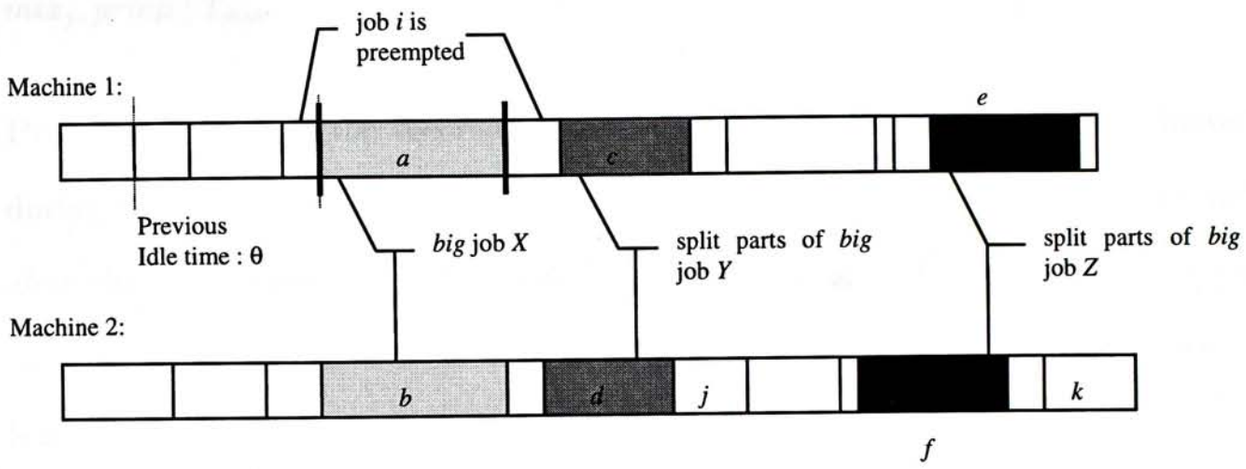

Figure 3.3: The gap θ is filled by a subsequence of jobs following job a, with length θ, and the remaining subsequence is shifted to start right after job a. Note that the remaining subsequence of jobs are unchanged before and after the adjustment

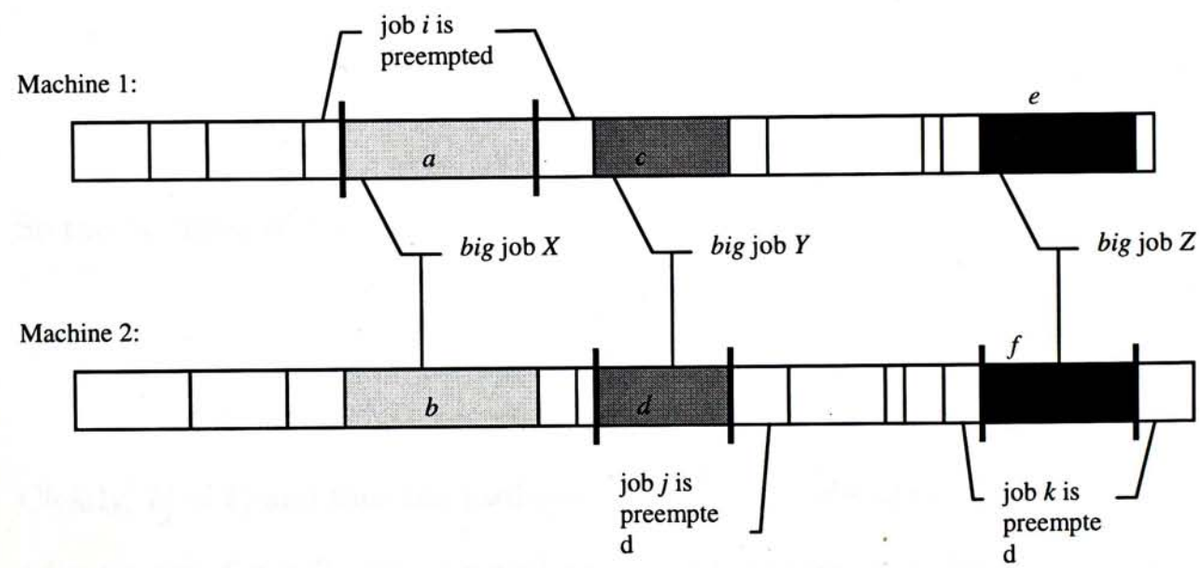

Figure 3.4: All *big* jobs processed simultaneously in both machines

Theorem 2 *The schedule obtained from Algorithm 2 is optimal to our problem,* $Q2 \mid mix_j, prmp \mid T_{\max}$.

Proof. We prove the theorem by considering the jobs which have been moved during an adjustment. First let us compare the tardiness of the *big* job before and after the adjustment. Let the *small* jobs split from *big* job j complete at c_j^1 on machine 1 and c_j^2 on machine 2. The tardiness of *big* job j, T_J, before the adjustment, is given by:

$$\begin{aligned} T_j &= \max\{0, c_j^1 - d_j, c_j^2 - d_j\} \\ &= \max\{0, \max\{c_j^1, c_j^2\} - d_j\}. \end{aligned} \tag{3.23}$$

After the adjustment, both *small* jobs completes at c_j, where:

$$c_j = \max\{c_j^1, c_j^2\}. \tag{3.24}$$

So the tardiness of *big* job j after adjustment is:

$$T_j' = \max\{0, c_j - d_j\}. \tag{3.25}$$

Clearly, $T_j' = T_j$ and thus the tardiness of all the *big* jobs are not increased after the adjustments. Secondly, let us consider the jobs following job a. They are shifted to be finished earlier and thus their tardiness are not increased, see figure 3. Furthermore, none of the completion times all other jobs have been changed, thus their tardiness

before and after adjustment remain the same. From the above, we can see that we obtain a feasible solution with an objective value that achieves a lower bound. Clearly, this is an optimal solution.□

Remarks

Since each pair of dedicated jobs split from the *big* jobs share the same due date after the first step of algorithm 2, they will be ordered consecutively after the EDD re-indexing. In applying Algorithm 2, we can modify, to save computing time, the recursive relationship for *big* job j as a combination of both split *small* jobs:

$$f_j(t_1, t_2) = \max\{f_{j-1}(t_1 - p_j^b, t_2 - p_j^b), \max\{0, t_1 - d_j, t_2 - d_j\}\}. \tag{3.26}$$

The back-tracking process may also been modified accordingly.

The Complexity of the Algorithm

There are 2 state values, t_1 and t_2, at each stage, The value of t_1 is bounded by P_j/v_1 (see equation 3.17), while the value of t_2 can be obtained from equation (??). If there are a total of n jobs, the optimal objective value is the minimum obtained at the nth stage:

$$\text{Optimal value } = \min_{t_1,t_2} \{f_n(t_1, t_2)\}. \tag{3.27}$$

It is not hard to see that the time complexity of the algorithm is $O(nP)$, where P is the total processing requirement of the n jobs.

Note that the complexity involved in each adjustment of the *big* jobs is of the order $O(n)$ and thus the time complexity of Algorithm 2 remains $O(nP)$.

Chapter 4

Extensions

In this Chapter we will discuss some related problems including some solvable cases, *set* problems, and the $k-$machine problem. We will tackle each case by applying the ideas we have used to solve our main problem.

4.1 Polynomially Solvable Cases $P2 \mid mix_j, prmp, p_j = p \mid T_{\max}$

We now consider the problem in which the processing requirements for all *small* jobs are the same, namely, $p_i = p$, and the two machines are identical, that is, $v_1 = v_2$. Without lost of generality, assume $v_1 = v_2 = 1$. Then, each job will have a processing time equals to its processing requirement. In this case, the optimal solution can be obtained efficiently by using a dynamic programming approach.

In solving this problem, we will use a similar idea to split each *big* job into a pair of dedicated *small* jobs with the same processing time and due date as the original *big* job. The problem again becomes a $P2 \mid prmp, p_j = p \mid T_{\max}$ problem. However, this time in the dynamic programming, we only need to consider the number of *small* jobs allocated to machine 1 at each stage. The algorithm is explained in details below.

4.1.1 Dynamic Programming

Again, we first re-index all *small* and *big* jobs in EDD order.

Stage : The problem is divided into *stages* $1, 2, ..., n$. Stage j corresponds to the decision of which machine job j should be assigned to.

State : At each stage, there are a number of various possible conditions which is represented by a *state* variable n_1, which is the number of *small* jobs allocated to machine 1.

Recursive Relationship : Let N_j be the number of *small* jobs in the set $\{1, 2, ..., j\}$. Define $f_j(n_1)$ as the minimum $T_{\max}$ given that we have assigned jobs $\{1, 2, ..., j\}$. Now we have the following recursive relationship:

Case 1: if job j is dedicated to machine 1, then

$$f_j(n_1) = \max\{f_{j-1}(n_1 - 1), \max\{t_1 - d_j, 0\}\} \tag{4.1}$$

Case 2: if job j is dedicated to machine 2, then

$$f_j(n_1) = \max\{f_{j-1}(n_1), \max\{t_2 - d_j, 0\}\} \tag{4.2}$$

Case 3: if job j is a *big* job, then

$$f_j(n_1) = \min\{\max\{t_1 - d_j, t_2 - d_j, 0\}, f_{j-1}(n_1)\} \tag{4.3}$$

Case 4: if job j is a *small* job and can be processed by either machine 1 or 2, then

$$\begin{aligned} f_j(n_1) \;=\; & \min\{\max\{f_{j-1}(n_1 - 1), \max\{t_1 - d_j, 0\}\}, \\ & \max\{f_{j-1}(n_1), \max\{t_2 - d_j, 0\}\}\} \end{aligned} \tag{4.4}$$

At each stage j, we have n_1 *small* jobs allocated to machine 1, and t_1 and t_2 are the latest completion times on machine 1 and 2, respectively. Note that the number of *small* jobs assigned in machine 2 at the jth stage is equal to $N_j - n_1$. Then the latest completion times on machines 1 and 2 are equal to the total processing times on both machines, namely,

$$t_1 = n_1 p + P_j^b \qquad \text{for machine 1, and} \tag{4.5}$$

$$t_2 = (N_j - n_1)p + P_j^b \qquad \text{for machine 2;} \tag{4.6}$$

where P_j^b is the sum of the processing times of the *big* jobs in $\{1, 2, ..., j\}$.

If job j is processed by machine i, then the tardiness for job j should be:

$$T_j = \max\{0, t_i - d_j\}. \tag{4.7}$$

If job j is processed by machine 1, then there will be $n_1 - 1$ *small* jobs processed by machine 1 at stage $j-1$, thus the tardiness for jobs $\{1, 2, ..., j-1\}$ is given by:

$$f_{j-1}(n_1 - 1). \tag{4.8}$$

Conversely, if job j is processed by machine 2, then there will be n_1 *small* jobs processed by machine 1 at stage $j-1$, thus the tardiness for jobs $\{1, 2, ..., j-1\}$ is given by:

$$f_{j-1}(n_1). \tag{4.9}$$

Ranges of n_1 : The range of n_1 is to cover all the possible states for each stage j. Clearly:

$$n_1 = 0, 1, ..., M_j, \tag{4.10}$$

where M_j is equal to the sum of the *small* jobs dedicated to machine 1 and the non-dedicated *small* jobs in $\{1, 2, ..., j\}$.

Initial Condition

The initial condition is given by:

$$f_0(n_1) = 0, \quad \text{for } n_1 = 0; \tag{4.11}$$

$$f_0(n_1) = \infty, \quad \forall n_1 \neq 0; \tag{4.12}$$

and

$$f_j(-1) = \infty. \tag{4.13}$$

Optimal Value : The optimal value, $T_{\max}$, is given by:

$$T_{\max} = \min_{n_1} \{f_n(n_1)\}. \tag{4.14}$$

Optimal Solution : Similar to our main problem we can obtain the optimal solution by back tracking. The optimal decision of the allocation of job j is given by the decision that gives the minimum objective value. At stage j, the optimal decision is that job J_j is allocated to machine 1 if:

$$f_j(n_1^o) = \min_{n_1} \{f_j(n_1)\} \tag{4.15}$$

and

$$f_j(n_1^o) = \max\{f_{j-1}(n_1 - 1), \max\{0, t_1 - d_j\}\}; \tag{4.16}$$

or it should be allocated to machine 2 if:

$$f_j(n_1^o) = \min_{n_1} \{f_j(n_1)\} \tag{4.17}$$

and

$$f_j(n_1^o) = \max\{f_{j-1}(n_1), \max\{0, t_2 - d_j\}\}., \tag{4.18}$$

where n^o is the value of n_1 such that $f_j(n_1)$ is minimized for all possible n_1.

The dynamic programming formulated above will solve $P2 \mid p_j = p \mid T_{\max}$ optimally. By introducing the adjustment as described in Chapter 3, the following algorithm solves the problem, $P2 \mid mix_j, prmp, p_j = p \mid T_{\max}$:

Algorithm 3

Step 1: Split each big job into a pair of dedicated small jobs, one dedicated to machine 1 and the other to machine 2. Both have the same processing times and due dates as the original big job.

Step 2: Re-index all jobs in Earliest Due Date (EDD) order.

Step 3: Set $f_0(n_1) = 0$ *for all* n_1.

Step 4: For $j = 1$ *to* n,

$n_1 = 0$ *to* M_j,

Compute the values of t_1 *and* t_2 :

$$t_1 = n_1 p + P_j^b;$$

$$t_2 = (N_j - n_1)p + P_j^b.$$

If job j a non-dedicated small job, *then:*

$$f_j(n_1) = \min\{\max\{f_{j-1}(n_1 - 1), \max\{t_1 - d_j, 0\}\}, \max\{f_{j-1}(n_1), \max\{t_2 - d_j, 0\}\}\}$$

else if job j is a small job dedicated to machine 1, then:

$$f_j(n_1) = \max\{f_{j-1}(n_1 - 1), \max\{t_1 - d_j, 0\}\}$$

else if job j is a small job dedicated to machine 2, then:

$$f_j(n_1) = \max\{f_{j-1}(n_1), \max\{t_2 - d_j, 0\}\}$$

else if job j is a big job, then:

$$f_j(n_1) = \min\{\max\{t_1 - d_j, t_2 - d_j, 0\}, f_{j-1}(n_1)\}$$

Step 5: Set the optimal value, $T_{\max}$, as the minimum of all f_n over different n_1:

$$T_{\max} = \min_{n_1} \{f_n(n_1)\}.$$

Step 6: Compute the state variable n_1^n for stage n such that:

$$\begin{aligned} T_{\max} &= \min_{n_1} \{f_n(n_1)\} \\ &= f_n(n_1^n). \end{aligned}$$

Step 7: Compute the values of t_1^n and t_2^n :

$$\begin{aligned} t_1^n &= n_1^n p + P_n^b; \\ t_2^n &= (N_n - n_1)p + P_n^b. \end{aligned}$$

Step 8: For $j = n$ to 1,(Generate the optimal schedule by back-tracking.)

If job j is a small job,

If job j is non-dedicated,

assign job j to machine 1 *when:*

$$f_j(n_1^j) = \max\{f_{j-1}(n_1^j - 1), \max\{0, t_1^j - d_j\}\}$$

else assign job j to machine 2 *if:*

$$f_j(n_1^j) = \max\{f_{j-1}(n_1^j), \max\{0, t_2^j - d_j\}\}$$

else assign job j to the prespecified machine.

Set the state variable n_1^{j-1}, t_1^{j-1} and t_2^{j-1} for stage $j - 1$:

If the job j is assigned to machine 1, then

$$\begin{aligned} n_1^{j-1} &= n_1^j - 1; \\ t_1^{j-1} &= t_1^j - p; \\ t_2^{j-1} &= t_2^j. \end{aligned}$$

else

$$\begin{aligned} n_1^{j-1} &= n_1^j; \\ t_1^{j-1} &= t_1^j; \\ t_2^{j-1} &= t_2^j - p. \end{aligned}$$

else if job j is a big job, assign job J_j as a pair of dedicated small jobs to machine 1 and 2 respectively.

Set the state variable n_1^{j-1}, t_1^{j-1} and t_2^{j-1} for stage $j-1$:

$$\begin{aligned} n_1^{j-1} &= n_1^j; \\ t_1^{j-1} &= t_1^j - p_j^b; \\ t_2^{j-1} &= t_2^j - p_j^b. \end{aligned}$$

where p_j^b is the processing time of job j.

Step 9: Adjust each pair of dedicated jobs corresponding to a same big job to start

and complete at the same time as described in Chapter 3 such that all big jobs occupy both machines at the same time.□

Time Complexity of the Algorithm

Note that n_1 has a maximum range of M_j which is bounded by n, the total number of *small* and *big* jobs. Also, similar to the case with different processing times, the adjustment for the *big* jobs needs a time $O(n)$. Thus, it is not hard to see that the algorithm has a time complexity $O(n^2)$.

4.2 Set Problem $P2/set_j, prmp/T_{\max}$

In this section, we consider the $P2/set_j, prmp/T_{\max}$ problem, which involves *set* jobs, as an extension for the $Q2/mix_j, prmp/T_{\max}$ problem. This problem is interesting itself because it includes two special cases, $P2/fix_j, prmp/T_{\max}$ and $P2/size_j, prmp/T_{\max}$.

In $Q2/set_j, prmp/T_{\max}$ problem, jobs can be classified into 7 categories:

1. *Small* jobs dedicated to machine 1 only.
2. *Small* jobs dedicated to machine 2 only.
3. *Big* jobs dedicated to both machines.
4. *Small* jobs that can be processed by either machine 1 or machine 2 only.
5. *Set* jobs that can be processed by either machine 1 only or by both machines.
6. *Set* jobs that can be processed by either machine 2 only or by both machines.

7. *Set* jobs that can be processed by either machine 1 only, or machine 2 only or by both machines.

For the *set* problem, we only consider the case where the processing speed, if a job is processed by two machines, is equal to (v_1+v_2), with v_1 and v_2 being the processing speeds of machines 1 and 2, respectively.

4.2.1 Processing times for *set* jobs

In our main problem, the jobs in categories 1, 2, 3 and 4 are considered. We now consider the processing times for jobs in categories 5, 6 and 7 when they are processed by different sets of machines. The formulation is analogous to that of jobs of categories 1 to 4 in our main problem.

First let us consider job j which belongs to category 5. If J_j has processing requirement p_j then when it is processed by machine 1, its processing time is given by:

$$p_{1j} = \frac{p_j}{v_1}. \tag{4.19}$$

Its processing time when processed by both machines is similar to that of a *big* job:

$$p_{12j} = \frac{p_j}{v_1 + v_2}. \tag{4.20}$$

Similarly, if J_j belongs to category 6 and is processed by machine 2, its processing time is given by:

$$p_{2j} = \frac{p_j}{v_2}. \tag{4.21}$$

And when it is processed by both machines, its processing time is:

$$p_{12j} = \frac{p_j}{v_1 + v_2}. \tag{4.22}$$

Furthermore, if J_j belongs to category 7, its processing time when processed by machine 1, p_{1j}, machine 2, p_{2j}, or by both machines, p_{12j}, are respectively given by:

$$\begin{aligned} p_{1j} &= \frac{p_j}{v_1}; \\ p_{2j} &= \frac{p_j}{v_2}; \\ p_{12j} &= \frac{p_j}{v_1 + v_2}. \end{aligned} \tag{4.23}$$

As we can see, if we put some processing times equal to infinite, category 7 covers all possible cases. For example, job j which belongs to category 1 has the following processing times:

$$\begin{aligned} p_{1j} &= \frac{p_j}{v_1}; \\ p_{2j} &= \infty; \\ p_{12j} &= \infty. \end{aligned} \tag{4.24}$$

If j belongs to category 3, the processing times are:

$$\begin{aligned} p_{1j} &= \infty; \\ p_{2j} &= \infty; \\ p_{12j} &= \frac{p_j}{v_1 + v_2}. \end{aligned} \tag{4.25}$$

Similarly for jobs from other categories, each job j will have a set of processing times, p_{1j}, p_{2j} and p_{12j}. Hence in the dynamic programming below, we just need to consider category 7.

4.2.2 Algorithm

The algorithm for solving our main problem, $Q2/mix_j, prmp/T_{\max}$ is applicable to give the optimal solution for the set problem. Note that the processing times for *set* jobs vary when they are processed by different set of machines. for job j, its processing time is p_{1j} and p_{2j} when processed by machine 1 and 2 respectively, and p_{i12} when processed by both machines simultaneously.

Consider a problem which is the same as the original *set* problem except that when jobs are assigned to be processed by both machines, they need not be processed simultaneously on both machines. We call this Problem 2. The *EDD* property also

applies to the optimal solution of Problem 2. Note that the optimal solution to Problem 2 is a lower bound to that of $Q2/set_j, prmp/T_{\max}$. We will first solve Problem 2, and then similar to the problem $Q2/mix_j, prmp/T_{\max}$, we can adjust each pair of dedicated *small* jobs split from the same *big* job such that all *big* jobs are processing simultaneously on both machines. The following Dynamic Programming will solve Problem 2 optimally.

Dynamic Programming

Re-index all *small*, *big* and *set* jobs in EDD order.

Determination of Processing Times and Processing Requirements

Compute the processing times of jobs in different categories.

Case 1, if job j is from category 1, then

$$\begin{aligned} p_{1j} &= \frac{p_j}{v_1}; \\ p_{2j} &= \infty; \\ p_{12j} &= \infty. \end{aligned} \tag{4.26}$$

Case 2, if job j is from category 2, then

$$\begin{aligned} p_{1j} &= \infty; \\ p_{2j} &= \frac{p_j}{v_2}; \\ p_{12j} &= \infty. \end{aligned} \tag{4.27}$$

Case 3, if job j is from category 3, then

$$\begin{aligned} p_{1j} &= \infty; \\ p_{2j} &= \infty; \\ p_{12j} &= \frac{p_j}{v_1 + v_2}. \end{aligned} \tag{4.28}$$

Case 4, if job j is from category 4, then

$$\begin{aligned} p_{1j} &= \frac{p_j}{v_1}; \\ p_{2j} &= \frac{p_j}{v_2}; \\ p_{12j} &= \infty. \end{aligned} \tag{4.29}$$

Case 5, if job j is from category 5, then

$$\begin{aligned} p_{1j} &= \frac{p_j}{v_1}; \\ p_{2j} &= \infty; \\ p_{12j} &= \frac{p_j}{v_1 + v_2}. \end{aligned} \tag{4.30}$$

Case 6, if job j is from category 6, then

$$\begin{aligned} p_{1j} &= \infty; \\ p_{2j} &= \frac{p_j}{v_2}; \\ p_{12j} &= \frac{p_j}{v_1 + v_2}. \end{aligned} \tag{4.31}$$

Case 7, if job i is from category 7, then

$$\begin{aligned} p_{1j} &= \frac{p_j}{v_1}; \\ p_{2j} &= \frac{p_j}{v_2}; \\ p_{12j} &= \frac{p_j}{v_1 + v_2}. \end{aligned} \tag{4.32}$$

Recursive Relationship

The recursive relationship is as follows:

$$\begin{aligned} f_j(t_1, t_2) = \min\{&\max\{f_{j-1}(t_1 - p_{1j}, t_2), \max\{0, t_1 - d_j\}\}, \\ &\max\{f_{j-1}(t_1, t_2 - p_{2j}), \max\{0, t_2 - d_j\}\}, \\ &\max\{f_{j-1}(t_1 - p_{12j}, t_2 - p_{12j}), \max\{0, t_1 - d_j, t_2 - d_j\}\}\} \end{aligned} \tag{4.33}$$

In addition, let:

$$f_j(t_1, t_2) = \infty \qquad \text{if } t_1 = -\infty \text{ or } t_2 = -\infty. \tag{4.34}$$

The algorithm for solving Problem 2 is given below.

Algorithm 4

Step 1: Re-index all jobs in Earliest Due Date (EDD) order.

Step 2: Define the processing times, p_{1j}, p_{2j} and p_{12j} *for all jobs as above..*

Step 3: Set $f_0(t_1, t_2)$ *to be zero when both* t_1 *and* t_2 *are equal to zero, and infinity*

otherwise.

Step 4: For $j = 1$ *to* n,

$t_1 = 0$ *to* P_j/v_1,

Set $t_2 = (P_j - t_1 \times v_1)/v_2$.

Compute:

$$\begin{aligned} f_j(t_1, t_2) \;=\; & \min\{\max\{f_{j-1}(t_1 - p_{1j}, t_2), \max\{0, t_1 - d_j\}\}, \\ & \max\{f_{j-1}(t_1, t_2 - p_{2j}), \max\{0, t_2 - d_j\}\}, \\ & \max\{f_{j-1}(t_1 - p_{12j}, t_2 - p_{12j}), \max\{0, t_1 - d_j, t_2 - d_j\}\}\} \end{aligned}$$

Step 5: Let:

$$T_{\max} = \min_{t_1, t_2} \{f_n(t_1, t_2)\}.$$

Step 6: Compute the state variables, t_1^n, t_2^n, *for stage* n *such that:*

$$\begin{aligned} T_{\max} &= \min_{t_1, t_2} \{f_n(t_1, t_2)\} \\ &= f_n(t_1^n, t_2^n) \end{aligned}$$

Step 7: For $j = n$ *to* 1, (Generate the optimal schedule by back-tracking.)

Assign job j *to machine* 1 *if:*

$$f_j(t_1^j, t_2^j) = \max\{f_{j-1}(t_1^j - p_{1j}, t_2^j), \max\{0, t_1^j - d_j\}\}.$$

Compute *the state variables,* t_1^{j-1}, t_2^{j-1},

$$\begin{aligned} t_1^{j-1} &= t_1^j - p_{1j}; \\ t_2^{j-1} &= t_2^j. \end{aligned}$$

Else assign job j *to machine* 2 *if:*

$$f_j(t_1^j, t_2^j) = \max\{f_{j-1}(t_1^j, t_2^j - p_{2j}), \max\{0, t_2^j - d_j\}\}.$$

Compute *the state variables,* t_1^{j-1}, t_2^{j-1},

$$\begin{aligned} t_1^{j-1} &= t_1^j; \\ t_2^{j-1} &= t_2^j - p_{2j}. \end{aligned}$$

Else assign job j *to both machines where:*

$$f_j(t_1^j, t_2^j) = \max\{f_{j-1}(t_1^j - p_{12j}, t_2^j - p_{12j}), \max\{0, t_1^j - d_j, t_2^j - d_j\}\}. \tag{4.35}$$

Compute *the state variables,* t_1^{j-1}, t_2^{j-1},

$$\begin{aligned} t_1^{j-1} &= t_1^j - p_{12j}; \\ t_2^{j-1} &= t_2^j - p_{12j}. \end{aligned}$$

Step 8: Adjust each pair of dedicated jobs corresponding to a same big jobs to

start and complete at the same time such that all big jobs occupy both machines at the same time.□

Time Complexity of the Algorithm

There are two state variables, t_1 and t_2, at each stage. Each t_1 has a maximum range of P_j/v_1, see equation (3.17), and t_2 can be obtained from equation (??). Thus, it can be seen that the algorithm has complexity $O(nP)$, where P is the total processing requirement of the n jobs.

4.3 $k-$Machine Problem with only two types of jobs

In this section, we will study $k-$machine problem with only two types of jobs, *small* and *big* jobs. *Small* jobs refer to regular jobs and can be dedicated or non-dedicated, while a *big* job must be processed by the k machines simultaneously. A pseudo-polynomial algorithm is given below to solve the problem.

When each *big* job is split into k *small* jobs dedicated to each of the k machines, the problem becomes a $Qk \mid\mid T_{\max}$ problem. We can obtain the optimal solution by

applying Algorithm 1 in chapter 3 with the recursive relationship modified as follows:

$$f_j(t_1, t_2, ..., t_k) = \min_{1 \le i \le k} \{\max\{f_{j-1}(t_1, t_2, ..., t_i - p_j/v_i, ..., t_k), \max\{0, t_i - d_j\}\}\} \quad (4.36)$$

However, in this case we have to ensure that each *big* job is processing simultaneously on k machines. The adjustment described in chapter 3 should be applied such that all split *small* jobs from the same *big* job have to start and complete at the same time. For each *big* job, there are at most $k-1$ adjustments, thus the time complexity needed by the adjustment for all the *big* jobs is $O(bk)$. In general, the algorithm for solving this $k-$machine problem has a time complexity $O(nP^{k-1})$.

The *big* jobs must be $k-$machine jobs in order that the algorithm above can be applied. This is because the adjustment cannot be applied to multi-processor jobs which occupy less than k machines. Two, or more, split jobs may compete for the same time slot on the same machine. For example, consider a problem where there are three machines and jobs involved may occupy one, two or three machines. Suppose that the multi-processor jobs are split into dedicated *small* jobs and Algorithm 1 is applied. Assume *big* jobs p and q are two-machine jobs. The pair of split jobs, p^1 and p^2, from job p are assigned to machine 1 and machine 2 while that from job q, job q^1 and job q^2, are assigned to machine 1 and machine 3. If both jobs q^2 and p^2 complete later than the other job from the same pair and they occupy the two overlapping time

slots, $[s_p, c_p)$ on machine 2 and $[s_q, c_q)$ on machine 3, where

$$s_p < s_q < c_p < c_q,$$

then job p^1 and job q^1 will complete for the time slot $[s_q, c_p)$ on machine 1, and therefore the adjustment cannot be applied.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

A new scheduling problem, $Q2 \mid mix_j, prmp \mid T_{\max}$ has been formulated and studied in this thesis. Its modelling has been motivated by the berth allocation problem in container terminals. We have shown that this is an NP-hard problem, but its solution can be obtained by an effective (pseudo-polynomial) algorithm we have proposed. Our algorithm is based on dynamic programming. The key idea is to make use of the possibility of preemption. Because of the possibility of preemption, we can first split a two-processor job into a pair of dedicated one-processor jobs, and then restore them into the original multi-processor job without changing the optimality of the solution obtained by a dynamic programming algorithm for the one-processor problem.

We have also extended the proposed approach to more general problems, including

the *set* problem and the $k-$machine problem. Nevertheless, for the *set* problem, we have been only able to solve the case where the processing speed when two machines are working together is equal to the sum of the speeds of the two machines. Note that in the *mixed* problem $Q2 \mid mix_j, prmp \mid T_{\max}$, we do not have to have this assumption, namely, we can deal with any processing speed when the two machines are combined to process a *big* job. This is also a reason why we have separated our treatment of the $Q2 \mid mix_j, prmp \mid T_{\max}$ problem from the *set* problem.

The $k-$machine problem has also been shown solvable in pseudo-polynomial time if all *big* jobs are $k-$machine jobs.

We have further proven that a special case, where all *small* jobs have equal processing requirements, is polynomially solvable.

5.2 Some Future Work

In our problem, there is a constraint that all one-machine jobs, or *small* jobs, must be completed by the same machine as long as they are assigned. Although preemption is allowed, no *small* jobs can be processed on another machine after preempted. The problem with this constraint relaxed is worth to be further studied.

Furthermore, it is interesting to solve the problem with other objective functions. Actually, we have attempted some other objectives, such as mean flow time or total tardiness. Unfortunately, in such cases, we cannot adjust the split jobs from the

big jobs without affecting the objective value and the results of this thesis may not hold. It would be interesting future research to develop other ideas to tackle objective functions.

Bibliography

[1] K.R. Baker, *Introduction to Sequencing and Scheduling,* Wiley, New York, 1974.

[2] R. Bellman, *Dynamic Programming*, Princeton University Press, 1962.

[3] L. Bianco, J. Blazewicz, P. Dell'Olmo and M. Drozdowski, *Preemptive Scheduling of Multiprocessor Tasks on the Dedicated Processor System Subject to Minimal Lateness*, Information Processing Letters 46 (1993) 109-113.

[4] L. Bianco, J. Blazewicz, P. Dell'Olmo and M. Drozdowski, *Scheduling Multiprocessor Tasks on a Dynamic Configuration of Dedicated Processors*, Annals of Operations Research in Scheduling 58 (1995) 493-517.

[5] L. Bianco, J. Blazewicz, P. Dell'Olmo and M. Drozdowski, *Linear Algorithms For Preemptive Scheduling of Multiprocessor Tasks Subject to Minimal Lateness*, Discrete Applied Mathematics 72 (1997) 25-46.

[6] L. Bianco, P. Dell'Olmo and M.G. Speranza, *Scheduling Independent Tasks with Multiple Modes*, Discrete Applied Mathematics 62 (1995) 35-50.

[7] J. Blazewicz, P. Dell'Olmo, M. Drozdowski and M.G. Speranza, *Scheduling Multiprocessor Tasks on Three Dedicated Processors*, Information Processing Letters 41 (1992) 275-280.

[8] J. Blazewicz, M. Drabowski and J. Weglarz, *Scheduling Multiprocessor Tasks to Minimize Schedule Length*, IEEE Transaction on Computers, c-35 (1986) 389-393.

[9] M. Drozdowski, *Scheduling Multiprocessor Tasks - An overview*, European Journal of Operational Research 94 (1996) 215-230.

[10] X. Cai, C.-Y. Lee and C.L. Li, *Minimizing Total Completion Time in Two-Processor task Systems with Prespecified Processor Allocations*, (1996), Naval Research Logistics 45 (1998) 231-242.

[11] Jianer Chen and C. -Y. Lee, *General Multiprocessor Scheduling Problems*, Chen, Department of Computer Science; Lee, Department of Industrial Engineering, Texas A&M University, (1997).

[12] Jianzhong Du and Joseph Y-T. Leung, *Complexity of Scheduling Parallel Task Systems*, Society for Industrial and Applied Mathematics Journal of Discrete Mathematics 2 No. 4 (1989) 473-487.

[13] S. French, *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*, Ellis Horwood Limited, 1982.

[14] 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.

[15] J.A. Hoogeveen, S.L. van de Velde and B. Veltman, *Complexity of Scheduling Multiprocessor Tasks with Prespecified Processor Allocations*, Discrete Applied Mathematics 55 (1994) 259-272.

[16] H. Krawczyk and M. Kubale, *An Approximation Algorithm for Diagnostic Test Scheduling in Multiprocessor Systems*, IEEE Transaction on Computers, c-34 (1985) 869-872.

[17] M. Kubale, *Preemptive Versus Nonpreemptive Scheduling of Biprocessor Tasks on Dedicated Processors*, European Journal of Operational Research 94 (1996) 242-251.

[18] C.-Y. Lee and X. Cai, *Scheduling Two-processor Tasks without Prespecified Processor Allocations*, 1996 submitted.

[19] C.-Y. Lee, Lei Lei, and Michael Pinedo, *Current Trends in Deterministic Scheduling*, Annals of Operations Research 70 (1997) 1-41.

[20] C. -L. Li, X. Cai and C. -Y. Lee, *Scheduling with Multiple-Job-on-One-Processor Pattern*, IIE Transactions on Scheduling and Logistics 30 (1998) 433-446.

[21] Zhen Liu and Eric Sanlaville, *Preemptive Scheduling with Variable Profile, Precedence Constraints and Due-dates*, Discrete Applied Mathematics 58 (1995) 253-280.

[22] M. Pinedo, *Scheduling : Theory, Algorithms, and Systems*, Prentice Hall, 1995.

[23] Jurgen Plehn, *Preemptive Scheduling of Independent Jobs with Release Times and Deadlines on a Hypercube*, Information Processing Letters 34 (1990) 161-166.