Some Problems in One-Operator Scheduling

by

Mohammed Fazle Baki

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Management Sciences

Waterloo, Ontario, Canada, 1999

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by examiners.

I understand that my thesis may be made electronically available to the public.

Acknowledgements

I would like to express my profound gratitude to my supervisor Professor Raymond G. Vickson for his constant guidance, encouragement and patience throughout the development of this thesis, for many hours of fruitful discussion, for financial support, and for giving me the opportunity to participate to several conferences.

I would like to thank Professors James Bookbinder, Elizabeth Jewkes, Anthony Vannelli and my external examiner Professor Yash Aneja for reviewing this thesis and for their helpful comments.

A special note of thanks is due to Professor Gordon Andrews who helped me so willingly with his expertise on gear manufacturing. Special thanks to Dr. Tim Nye for some fruitful discussion and valuable information. Special thanks to Dr. Haldun Süral who has shown so much interest in some of the problems I have worked on and offered some excellent suggestions.

I wish to thank Lynne Wight for various helps she offered to me during writing of this thesis. Thanks to Stephen Carr who has helped me with so much enthusiasm on various aspects of document preparation.

Thanks to the Natural Sciences and Engineering Research Council of Canada, Ontario Ministry of Training, Waterloo Management of Integrated Manufacturing Systems (WATMIMS) research group, and the University of Waterloo for their financial support throughout my graduate studies.

Thanks to Professor S.N. Kabadi who supervised my MBA thesis and taught me the fundamentals of combinatorial optimization. Thanks for his encouragement and inspiration.

Thanks to my mother, my sister and my wife for being so supportive throughout and for all their encouragement, inspiration and moral support. Special thanks to my baby, Shaumik for lots of fun together.

Dedication

In memory of

my father late Mohammed Solaiman Ali and

my brother late Mohammed Fazle Bari

Abstract

A flexible workforce or a versatile machine is employed to perform various types of operations. Often these resources are associated with setups. Whenever a worker or machine switches from processing one type of operation to another a setup time may be required although several operations of a same type can be processed in succession after a single setup.

The presence of setups gives rise to the problem of choosing batch sizes that are neither too large nor too small. In the last one and a half decade, many researchers have addressed the problem of scheduling with batching. A majority of articles assumes that there is only one type of scarce resource, which is typically machine. Often there can be two scarce resources such as a worker and a machine or a machine and a tool.

We propose a resource constrained scheduling model with a single operator and two or more machines. Whenever the operator changes machine, a setup time is required that may be sequence dependent or sequence independent. We consider the two cases of an open shop and a flow shop. In the open shop case, the order in which a job visits the machines is unrestricted. In the flow shop case, every job must visit the machines in the same order. We consider various scheduling objectives.

For variable number of machines, many cases are intractable. We discuss some dominance properties that narrow down the search for an optimal schedule. We

present a dynamic programming approach which solves a large number of cases. The running time of the dynamic program is polynomial for a fixed number of machines.

For the case of two machines, we show that the dominance properties have a nice interpretation. We develop some algorithms and justify their use by establishing running times, comparing the running times with those of the existing algorithms, and testing the performance of the algorithms.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Scheduling with Flexible Resources and Setups

As Baker [11] defines it, *scheduling is the allocation of resources over time to perform a collection of tasks.* In the manufacturing context, workers, machines and tools are some examples of resources. Tasks include *operations* that bring some physical changes to materials in order to eventually manufacture products. Tasks also include *setups* that do not bring such physical changes but are essential to carry out operations. Examples of setups include walking to reach the workplace, obtaining tools, positioning the work-in-process material, returning tools, cleaning, setting the required jigs and fixtures, adjusting tools, and inspecting material.

The length of time over which a scheduling decision has an effect is usually short, such as hours, days or months. Like any other short-term decisions, scheduling decisions are made by first-level management at a high level of detail. Intermediate-term

decisions made by mid-level management, and long-term decisions made by senior-level management, may constrain scheduling choices. For instance, mid-level management may be responsible for the acquisition of capital equipment and planning the workforce level, while senior-level management may be responsible for the choice of production lines and technology.

Recent trends in manufacturing include: (i) increasing demand toward customized products; (ii) increasing competition for market share from both domestic and international manufacturers; (iii) changing manufacturing technology; and (iv) changing customer needs and shorter product life cycles.

To respond to this trend, manufacturers set various goals including continual development of new products, reduction in the cost of production, and improvement in quality and service. Flexible resources such as a cross-trained or multiskilled workforce and versatile machines help the manufacturers to achieve these goals.

A flexible workforce reduces the impact of uncertainties such as product mix changes and demand changes. Many companies (Kher et al. [67]) such as Frito Lay and General Motors of USA use a flexible workforce. Frendall et al. [39] discuss a scheduling problem in a large toolroom of an automobile plant in central Michigan, where there is more machine capacity than labor capacity, in which most operators can operate at least two workstations, and in which the operators are cross-trained to operate every workstation in the toolroom. Liao and Lin [82] discuss a case study of a

manufacturing company producing sewing machine parts with a total of 80 machines and 17 cross-trained workers. Some of the machines require the full-time presence of an operator for as long as the machine runs.

Like a flexible workforce, a versatile machine such as a Computer Numerical Controlled (CNC) machine also reduces the impact of uncertainties. The aim of a CNC machine is to achieve the efficiency of automated mass production and, yet, be able to handle some variations among the products. For these reasons, CNC machines are widely used (Stecke [110]) in the metal-working industry, where products are manufactured in batches, as the variety is neither too low nor too high.

When a worker or machine switches from processing one type of operation to another, a setup time or cost may be incurred. All the operations processed in a single setup comprise a *batch of operations*. Larger batches are attractive because of fewer setups, less loss of production time, higher utilization of resources, more throughput and less time required to process all the operations. On the other hand, a smaller batch may prevent an important operation from waiting for a prolonged time for a different setup. Smaller batches may also reduce storage space requirements, the amount of capital tied up in inventory, the average length of time required between the receipt of an order and its release, etc.

The presence of setup times, therefore, gives rise to the problem of choosing batch sizes that are neither too large nor too small. In the last one and a half decades, there

has been a lot of interest in the scheduling problem with setups. Allahverdi et al. [5] and Potts and Kovalyov [95] present recent reviews on the scheduling problem with setups.

A collection of operations on a single product is called a *job.* The scheduling literature thoroughly considers the scheduling problem with setups with the assumption that there is only one type of scarce resource, which is typically a machine. However, often there can be two scarce resources, such as a worker and a machine or a machine and a tool. The presence of two scarce resources is recognized in the area of resource-constrained scheduling. We shall now propose a resource-constrained scheduling model with setups. Equivalent problems and models will be discussed after the statement of the problem and the presentation of some notation and definitions.

1.2 One-Operator Scheduling

Suppose that a single operator has to perform some n jobs on some m machines. Each job j requires $n_{ij} \geq 1$ operation on machine i. The processing time of job j at the k-th visit on machine i is t_{ijk}. If $n_{ij} = 1 \ \forall i, j$ we shall omit the suffix k and denote the processing time of job j on machine i by t_{ij}. A setup time $s_{ii'}$ is required each time the operator switches from machine i to i'. The initial setup time is s_{0i} on machine i. If, for each machine i' we have $s_{ii'} = s_{i'} \ \forall i$, then the setup times are *sequence independent.* We assume that several jobs can be processed in succession

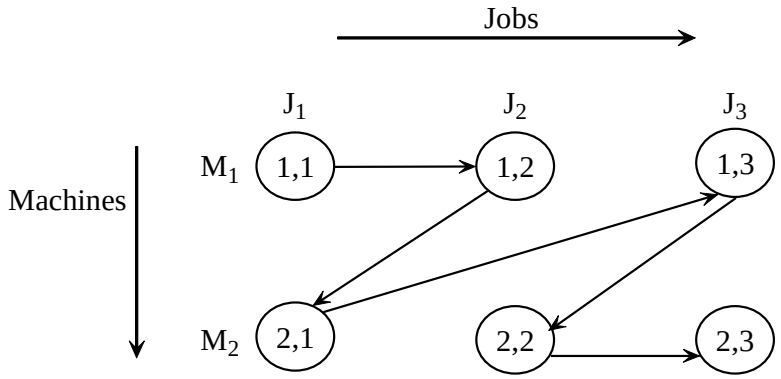

Figure 1.1: An illustration of sequences of operations

after a single setup. The operator can perform only one operation at a time and cannot perform any operations while a setup is in progress. Associated with every job j, there is a *due date* d_j and a *weight* w_j, which measures the importance of job j relative to the other jobs in the system.

As an example of the operator's scheduling problem, suppose that there are two machines and three jobs, each of which has exactly one operation on each machine. The operator may first do all jobs in some order on machine 1, and then do all jobs in some order on machine 2. An alternative sequence is shown in Figure 1.1. Every node $v = (i, j)$ in Figure 1.1 represents the operation of job j on machine i and every arc (v, v') represents the fact that the operation corresponding to node v is followed immediately by the operation corresponding to node v'. Hence, Figure 1.1 illustrates a schedule in which the operator processes jobs 1 and 2 on machine 1, then job 1 on machine 2, then job 3 on machine 1 and, finally, jobs 2 and 3 on machine 2.

Throughout we assume that (i) t_{ijk}, $s_{ii'}$, d_j, w_j are non-negative integers; (ii) setup times follow the triangle inequality condition, $s_{i_1 i_2} + s_{i_2 i_3} \geq s_{i_1 i_3}$ for every set of three distinct machines i_1, i_2 and i_3; (iii) jobs, machines and the operator are available at time zero and remain available during the entire scheduling period; and (iv) operations and setups are non-preemptive, meaning that once an operation or setup is started it must be completed without interruption.

We consider the two cases of an *open shop* and a *flow shop*. In the open shop case, the order in which a job visits the machines is unrestricted. In the flow shop case, every job must visit the machines in the same order, which we assume to be first on machine 1, second on machine 2, and so on.

1.2.1 Scheduling Objectives

The ultimate aim of any scheduler is to develop a feasible schedule that is optimal with respect to some objective. The first step in solving a scheduling problem is thus to define the scheduling objective. However, choosing a scheduling objective is, itself, a challenging problem. Often, schedulers must deal with many conflicting objectives. Mellor [83] lists a total of 27 objectives that may be important in a manufacturing environment. These objectives can be classified primarily as those that are *regular* and those that are not.

The *completion time* C_j of job j is the epoch at which its last operation is finished. A *regular objective* is one which is non-decreasing in completion times. Precisely,

a regular objective δ is a function such that $C_j \leq C'_j \ \forall j \Rightarrow \delta(C_1, C_2, ..., C_n) \leq \delta(C'_1, C'_2, ...C'_n)$.

The objectives can be further grouped into three broad categories: (i) efficient utilization of resources; (ii) average length of time spent on the shop floor; and (iii) conformance to prescribed deadlines.

We discuss in Section 1.4 that a one-operator scheduling problem may arise in various different contexts. Scheduling objectives may differ considerably from one context to another. Hence, we choose to explore various cases of the one-operator scheduling problem. In each case we consider a single regular objective.

Before we may list the objectives, we need some definitions. The completion time of the last job processed is called the *makespan* and is denoted by $C_{\max}$. The *lateness* L_j of job j is defined as $C_j - d_j$. The *tardiness* T_j of job j is defined as $\max\{0, L_j\}$. The *unit penalty* U_j of job j is defined as $U_j = 1$, if $C_j > d_j$, and $U_j = 0$, otherwise. Thus, $\sum U_j$ represents the *number of tardy jobs*.

We consider the following objectives: (i) makespan, $C_{\max}$; (ii) maximum lateness, $L_{\max}$; (iii) total completion time, $\sum C_j$; (iv) weighted completion time, $\sum w_j C_j$; (v) number of tardy jobs, $\sum U_j$; and (vi) weighted number of tardy jobs, $\sum w_j U_j$.

Rinnooy Kan [99] develops some equivalence relationships that exist among scheduling objectives. He shows that minimizing makespan is equivalent to maximizing utilization of resources, minimizing loss of production time due to setups and maximizing

throughput. Minimizing total (weighted) completion time is related to minimizing mean manufacturing lead time (which is the length of time between receipt of an order and its delivery), and minimizing work-in-process inventory. Minimizing maximum lateness and (weighted) number of tardy jobs is related to the level of conformance to prescribed deadlines and hence to customer service.

1.3 Problem Classification

Following Graham et al. [51] and a number of subsequent books and papers, we describe a scheduling problem by a triplet $\alpha|\beta|\gamma$.

The α field describes the operator and machine environment. We let $\alpha = \alpha_1\alpha_2\alpha_3$. Subfield α_1 indicates the number of operators. In the case of one-operator problems, $\alpha_1 = 1$. Subfield $\alpha_2 \in \{O, F\}$, where "O" indicates the open shop environment and "F" indicates the flow shop environment. Subfield $\alpha_3 = m$, where m is the number of machines. In a classical machine scheduling problem, such as a single machine, open shop or flow shop problem, the operator is not considered as a scarce resource and, therefore, there will be no entry in the subfield α_1. In such cases $\alpha = Om$ for an m-machine open shop problem and $\alpha = Fm$ for an m-machine flow shop problem. In the case of a single machine problem, the α field contains a single entry, 1.

The β field provides the details of processing characteristics and constraints. It is a standard practice in the scheduling literature to mark the presence of setup times

with an entry in the β field. Following this practice, we enter s_i in the β field if the setup times are *sequence independent* and $s_{ii'}$ if the setup times are *sequence dependent*. Two special terms used in the β field are as follows: (i) *apt: processing times are agreeable* in the sense that $j < j' \Rightarrow t_{ij} \leq t_{ij'} \forall i$; and (ii) *aptw: processing times and weights are agreeable* in the sense that $j < j' \Rightarrow t_{ij} \leq t_{ij'}\ \forall i$ and $w_j \geq w_{j'}$. Any other entries in the β field will be self explanatory.

The γ field contains the objective to be minimized.

1.4 Equivalent Problems and Models

In this section we shall discuss some problems that can be modelled as one-operator scheduling problems. Also, we shall discuss some equivalent models which appear in the scheduling literature in seemingly unrelated contexts.

Products may often require more than one component and the production of all the components may be carried out using a single manufacturing facility, worker, machine, or workstation. If the products are delivered after all the components are produced, each product can be viewed as a *job* and the production of each component can be viewed as an *operation*. In another context, customer orders may contain a number of products and all the products may be produced in a single production facility. If the customer orders are delivered after all the products are produced, each customer order can be viewed as a *job* and the production of each product can be

viewed as an *operation*. We shall now discuss such cases in further detail.

1.4.1 Gears On A Hobbing Machine

A hobbing machine is used for manufacturing a wide range of parts, including gears. The American Gear Manufacturers Association (AGMA) includes more than 200 manufacturers of gears, and most of them use hobbing machines.

A typical automotive power transmission gear unit usually contains a reverse gear and two to five forward gears, depending on the number of speed levels desired. Each gear set comprises three components: a sun gear, a planetary gear and a ring gear. Furthermore, all of these components may be produced on a single hobbing machine. However, before a batch of a particular component can be produced on the machine, the machine has to be set up. For a Computer Numerical Control (CNC) hobbing machine, the setup operation may take about two hours [9]. This includes changing the tooling, fixtures or pallets in the carousel, and end effectors on the load and unload devices. For a manual hobbing machine, the setup operation is typically about two and a half hours [9].

Apart from the shorter setup time required, CNC hobbing machines have the following additional advantages [9] over manual hobbing machines: (i) closer tolerances; (ii) more uniform production from part-to-part and lot-to-lot; and (iii) a faster machining cycle. For example, a manual hobbing machine may produce 15 parts per hour, but a CNC hobbing machine may produce 28 parts per hour. For this reason,

using CNC hobbing machines instead of manual hobbing machines seems to be a standard practice today [9].

The price of a CNC hobbing machine is often cited in the range of \$250,000–\$500,000 (US); see [8, 60, 116]. Such a high price is a barrier to using a large number of CNC hobbing machines in order to minimize loss of production time due to setups. For this reason, a manufacturer may sometimes have a single CNC hobbing machine. This is especially true for a manufacturer who is a new entrant in the gear manufacturing industry or for a manufacturer who is currently purchasing gears from an outside vendor and wants to bring the gear manufacturing in-house. For example, the Ann Arbor Machine Co. [10, 92] spent 1 million (\$ US) in the course of a year for gears supplied by external vendors. The company bought a single CNC hobbing machine when it decided to manufacture gears in-house, in order to reduce the turnaround time from as high as two months to only a few hours.

In general, if a manufacturer produces a product that requires m types of gears (so that the hobbing machine has to be set up m times for each product), we have a one-operator, m-machine open shop problem. If an automotive power transmission gear unit contains k gears (typically, $2 \leq k \leq 6$), each containing a sun gear, a planetary gear and a ring gear, then $m = 3k$. However, there exist other instances which allow us to consider problems with $m = 2$. For example, a product with just two types of gears would be a gear motor that requires a worm gear and a follower

gear.

One limitation of the model is that if a gear is required to sustain a high power and high torque, the steel of the gear may be required to have a hardness which the hobbing machine cannot cut. In this case, the cutting operation is carried out on a soft steel, and the gears cut on a hobbing machine are subjected to a hardening process. The component gears are usually moved to the hardening process in batches. Another limitation of the model is that some of the gears may be so big that they do not fit on a hobbing machine. A shaping machine is usually used for producing such a big gear.

These considerations, however, present no further difficulties [6] for modeling the problem of scheduling automotive power transmission gear units and gear motors, which usually require gears with small size, high speed and low torque. The required gears usually fit on a hobbing machine and the machine is usually capable of cutting the steel with the required hardness.

1.4.2 Partpieces with Two Operation Types

Cheng and Wang [26], Lee and Mirchandani [78] and Pan and Chen [90] discuss another problem of scheduling partpieces on versatile machines. If the production of a partpiece requires a tool that is not resident in the finite capacity tool magazine of the machine, then some or all of the tools in the tool magazine must be replaced. For example, suppose that there are three types of partpieces, A, B and C which require

tool sets {1,2,5}, {2,3,6} and {4,5,6}, respectively. Suppose that the tool magazine may contain at most 4 tools. If operation type X corresponds to tool set {1,2,3,4} and operation type Y corresponds to tool set {1,2,5,6}, then parts A, B and C all require both operation types. Lee and Mirchandani [78] assume that: (i) the versatile machine can perform all operations of the two types X and Y; (ii) a constant setup time is required to switch between operation types; and (iii) each partpiece requires an operation of type X, followed by an operation of type Y. Cheng and Wang [26], Lee and Mirchandani [78] and Pan and Chen [90] consider the problem with two identical versatile machines. However, if for any reason we have to schedule jobs on a single versatile machine, we obtain a one-operator, two-machine flow shop problem. In this formulation the versatile machine becomes the "operator," and the operation types become the two "machines."

1.4.3 Products with Two Types of Components or Subassemblies

Baker [12], Aneja and Singh [7], Ding [34], Vickson et al., [117], Sung and Park [111] and Rana and Singh [98] discuss a two-stage manufacturing problem. The first stage is production and the second stage is assembly. The production stage is capacity constrained because production is carried out on a single manufacturing facility. The assembly stage is not capacity constrained. Therefore, it is important to consider the scheduling problem in the production stage. Primarily, two types of components are

produced in the production stage. One type of component is common to all products and the other types of components are unique to each product.

The components are produced in batches , and a setup is required to produce each batch of a given component. The setup times are independent of sequence, and there is no limit on the size of any batch. Setup times for the unique components are imbedded in the processing times. Hence, it suffices to assume setup times only for the common components.

There can be two distinct assumptions regarding the availability of completed components; see (Santos and Magazine [107] and Dobson et al. [35]). Baker [12], Aneja and Singh [7] and Ding [34] assume that neither the unique nor the common components may be shipped to assembly until all unique and all common components in a production batch are completed. This way of moving products is called *batch availability* (Santos and Magazine [107]). On the other hand, Vickson et al. [117] and Sung and Park [111] assume that both the unique and common components are shipped to the assembly operation as soon as both have been completed on the production facility. This way of moving products is called *item availability* (Santos and Magazine [107]).

For the case of batch availability and total completion time objective, Baker [12] shows that the batch of common components is always processed prior to its corresponding unique components. An interchange argument shows that this statement

can be extended to any regular objective. But then, for any regular objective the problem with batch availability can be modelled as a one-operator two-machine flow shop problem: the first machine produces common components and the second machine produces unique components. On the other hand, the problem with item availability can be modelled as a one-operator two-machine open shop problem: one machine produces common components and the other machine produces unique components.

A simpler case of producing components was considered by Coffman et al. [30]. Consider a manufacturing system where products are composed of different subassemblies and a single machine makes the subassemblies of each type. All the subassemblies of a particular product can be produced in any order. However, a fixed setup time is needed whenever the machine changes over from one type of subassembly to another. Coffman et al., point out an application where a machine inserts components into circuit boards of different types, which are then assembled into kits. Coffman et al. consider the production of a single product that is composed of two subassembly types. This problem can be modelled as a one-operator two-machine open shop problem with identical jobs.

Each of the above articles except Ding [34] assumes *agreeable processing times*. Recall that the processing times are agreeable if there exists a labelling such that $j < j' \Rightarrow t_{ij} \leq t_{ij'} \forall i$. Vickson et al. assume identical processing times for all common components. Coffman et al. [31] assume identical processing times for all products

and identical setup times for both subassemblies. Ding [34] does not consider any restriction on processing times.

With the exception of Aneja and Singh [7], each of the above articles considers the case of two components. Aneja and Singh [7] point out that m'-common components cannot be treated as a single component, and provide an algorithm for the case of m'-common components and one unique component.

All the above articles consider minimizing total completion time. Rana and Singh [98] consider multiple objectives including total completion time and makespan.

Gim and Han [49], and Cheng and Wang [27] consider the integrated problem of scheduling the production and assembly stages. Gim and Han [49] consider m-components with the objective of minimizing total production cost including work-in-process inventory cost, total setup cost and inventory holding cost of the final product. Cheng and Wang [27] consider one common component and one unique component with the objective of minimizing makespan.

1.4.4 Customer Orders

Julien [63] and Julien and Magazine [64] present a model for scheduling customer orders. They consider a manufacturing system comprising two stages. The first stage is production, which represents either a product fabrication stage in a make-to-order environment or a product assembly stage in an assemble-to-order environment. The second stage is distribution.

In the first stage m types of products can be produced. Raw materials, components, and subassemblies needed to fabricate or assemble the products are always available in sufficient quantities. A product-dependent setup time s_i is incurred immediately before a batch of product i can be produced. Every customer needs a certain number of items of each product type.

The second stage, distribution, is not capacity constrained. Julien and Magazine [64] point out two motivations for such an assumption. First, there may always exist a sufficient delivery capacity. Second, the manufacturer may use an f.o.b. (free on board) factory pricing policy which stipulates that customers take ownership of their completed order at the factory and be responsible for transportation beyond the factory. As the distribution stage is not capacity constrained, it is sufficient to consider the scheduling problem that arises in the production stage. Julien and Magazine [64] consider many scheduling objectives, each of which is regular.

If we impose an additional restriction that every customer order requires at least one item of each product, then the scheduling problem in the production stage is a one-operator, m-machine open shop problem in which the production facility becomes the "operator," product types become the "machines" and customer orders become the "jobs". Julien and Magazine [64] considers several cases, including one with $m = 2$ and the restriction that every customer order requires at least one item of each product.

1.4.5 Aluminum Extrusion Facility

Bedworth and Bailey [13] discuss a problem that occurred in an aluminum extrusion facility. The plant has 10 extrusion presses of differing sizes and capabilities. The presses run in parallel. Different-size aluminum billets are extruded into 287 different shapes. Extrusion dies fit into specific die carriers. Machines are set up with one carrier at a time. The carriers can be changed as needed.

Orders for extrusions are booked in different quantities and have widely varying delivery lead-times. Marketing practices tend to offer faster deliveries to key customers. In general, Marketing would like to reduce manufacturing lead times to everyone in order to gain a competitive edge. However, Manufacturing tries to minimize layoffs, hiring, and overtime and, therefore, would like to increase the manufacturing lead times.

When customers book orders, they typically request several different extrusion shapes and sizes. It is desirable to have all of these products ready at the same time so that they can all be shipped at once.

The dies are designed to fit into specific die carriers and each machine is limited in the carriers it can accommodate. When a press is changed over from one operation to another, the effort depends on whether the carrier needs to be changed along with the die. Setups that require only die be changed have a standard time of one hour, but setups that require the carrier be changed have a standard time of four hours.

A survey identified that there were 27 different die carriers. Thus, all the products were placed in one of 27 groups. A study of past orders for products in each group indicated that for three groups, orders arrived faster than they could be produced on one press. Thus, it was decided to devote three presses exclusively and continuously to these three die carriers. The sequencing of the operations on each of the high-volume carrier groups was accomplished according to a modification of the algorithm of Moore and Hodgson (Moore [86]) for minimizing total number of tardy jobs on a single machine. The remaining 24 carriers and seven presses would be scheduled by a different rule.

Consider the problem of sequencing the operations on a high-volume carrier group. Since the machine is devoted to a die carrier, no setup for die carriers is needed. However, every time the die is changed a setup is required. Since customers book orders with several different extrusion shapes and sizes, it is reasonable to assume that every customer order requires that every die be used *at least once*. In fact, this assumption is used by Gupta et al. [55]. But then, the problem can be modelled as a one-operator (i.e., the high-volume die carrier and the associated extrusion press) m-machine (i.e., m dies) open shop problem.

1.4.6 Single-Machine, Multi-Operation Problem with Setup Times

Santos [106] considers a problem which he calls a single machine, multi-operation problem. Consider a single versatile machine and a product with m-operations. All the operations of some n items of the product are to be processed on the machine. The operations of each item are performed in a given order. Every time the machine switches to the i-th operation, a setup time s_i is required. Santos [106] considers the objective of minimizing total completion time. The only difference between his problem and our $1Fm|s_i, t_{ij} = t_i|\sum C_j$ problem is that he assumes batch availability of all the operations.

Recently, Gerodimos et al. [45, 46] consider the single machine, multi-operation problem with item availability assumption and various regular objectives. They assume that each product j comprises one operation of each type $i = 1, 2, .., m$. The processing time of product j for operation type i is t_{ij}. Product j requires n_{ij} operations of type i, where $n_{ij} \in \{0, 1\}$. For each product the items can be processed in any order. The restricted case of their problem with $n_{ij} = 1$ is equivalent to $1Om|s_i|\delta$.

1.4.7 Summary of Equivalences

As we have seen above, one-operator scheduling problem may arise in many different contexts. In Table 1.1 we summarize how we may view the problems, or some restricted version of the problems, as a one-operator problem. In Section 3.5 we shall

show that a well-studied single machine problem is a special case of the one-operator scheduling problem.

Problem	**Equivalent One-Operator Model**
Automotive transmission gears on a hobbing machine	$1Om\|s_i\|\delta$
Gear motors with worm gear and follower gear on a hobbing machine	$1O2\|s_i\|\delta$
Partpieces with two operation types on a single versatile machine	$1F2\|s_i\|C_{\max}$
Common and unique components, Baker [12]	$1F2\|s_i, s_2 = 0, apt\|\sum C_j$
Common and unique components, Ding [34]	$1F2\|s_i, s_2 = 0\|\sum C_j$
Common and unique components, Vickson et al. [117]	$1O2\|s_i, s_2 = 0, t_{1j} = t_1\|\sum C_j$
Common and unique components, Sung and Park [111]	$1O2\|s_i, s_2 = 0, apt\|\sum C_j$
Two subassemblies, Coffman et al. [30]	$1O2\|s_i, s_1 = s_2, t_{ij} = t_i\|\sum C_j$
Customer order scheduling, $n_{ij} \geq 1$, Julien [63]	$1Om\|s_i\|\delta$
Customer order scheduling, Julien and Magazine [64]	$1O2\|s_i, \textit{fixed sequence}\|\sum C_j$
Aluminum extrusion press, a machine devoted to a high-volume die carrier, $n_{ij} \geq 1$,	$1Om\|s_i\|\delta$
Single-machine multi-operation problem, $n_{ij} = 1$, Gerodimos et al. [45, 46]	$1Om\|s_i\|\delta$

Table 1.1: Summary of Equivalences

1.5 Theory of Computational Complexity

The scheduling problem belongs to the area of *combinatorial optimization*. In a combinatorial optimization problem we have to choose the best from a finite number of feasible solutions. For example, in a one-operator scheduling problem with m machines and n jobs, there are mn operations, and these can be carried out in $(mn)!$ ways. The finiteness of the solution set immediately gives an *explicit enumeration scheme*: generate all the feasible solutions, compute the objective functions and choose the best one. However, such a scheme leads to what is known as *combinatorial explosion*: the computational burden grows exponentially with the number of machines and/or jobs.

Edmonds [36] introduced the concept of a *good algorithm*. He called an algorithm good if the number of steps required to solve a problem can be expressed as a polynomial function of the length of the input data. Inputs are usually encoded *in binary* notation; e.g., the integer 5 is represented as 111. In binary encoding a positive integer a has a length of $\lfloor \log_2 a \rfloor + 1$.

The *running time* of an algorithm is expressed as a function of the length of the input and provides the rate of growth of the number of elementary steps required. If the length of the input is measured by x and the running time of an algorithm is $O(f(x))$, this means that there exist constants c and x_0 such that the number of elementary steps required by the algorithm is at most $cf(x)$ for any $x \geq x_0$.

The algorithm is a *polynomial-time algorithm* if $f(x)$ is a polynomial function. Any algorithm whose running time cannot be so bounded is called an *exponential-time algorithm.* The *space requirement* of an algorithm is similarly defined.

If an algorithm for a problem with n jobs requires $(6n^2 + 4n + 3)$ steps, we ignore the term $(4n + 3)$ and say that the running time is $O(n^2)$ because, for $c = 13$ and $n_0 = 1$ the number of steps $(6n^2 + 4n + 3) \leq cn^2$ for $n \geq n_0$. Similarly, if an algorithm for a problem with n jobs requires $(35.2^n + n^3)$ steps, we say that the running time is $O(2^n)$ because, for $c = 39$ and $n_0 = 1$ the number of steps $(35.2^n + n^3) \leq c2^n$ for $n \geq n_0$. The former algorithm is polynomial-time and the latter algorithm is exponential-time.

A problem is *polynomially solvable* if an optimal solution is obtained by a polynomial-time algorithm. Many combinatorial problems, called *intractable problems,* are not known to be polynomially solvable. The *computational complexity* paradigm provides a methodology to demonstrate intractability of a problem. Specifically, when investigating a new problem, it is often possible to show that if the problem is polynomially solvable, then many other intractable problems are polynomially solvable. For an introduction to the paradigm, we refer to the seminal works by Cook [33], Karp [66] and the textbook by Garey and Johnson [41]. Below, we discuss some basic concepts.

The theory is discussed in the context of *decision* problems rather than *optimization* problems. A decision problem is a question answered by 'yes' or 'no'. An

optimization problem can be solved by solving a finite number of decision problems. For a scheduling problem, a decision problem is defined with some *threshold* value $\mathcal{D}$ as follows: does there exist a schedule σ with objective function $\delta(\sigma) \leq \mathcal{D}$? The scheduling problem can thus be solved by repeatedly adjusting the threshold value $\mathcal{D}$ in a binary search over an appropriate interval for $\mathcal{D}$.

For the above decision problem if we have a schedule σ with $\delta(\sigma) \leq \mathcal{D}$, we say that schedule σ *certifies* that the answer is 'yes'. For many decision problems, certifying a 'yes' answer may require a small piece of information and a few steps of computation. For example, in the context of one-operator scheduling, a 'yes' answer to the above decision problem can be certified very easily. It requires at most $O(mn)$ time to define schedule σ, compute job completion times and thus verify whether σ certifies that the answer is 'yes'.

In general, a decision problem is called *non-deterministic polynomial* if there exists a polynomial-time algorithm that accepts an input with a polynomial length and verifies whether the input certifies that the answer is 'yes'.

The class of non-deterministic polynomial problems is denoted by $\mathcal{NP}$. For the polynomially solvable decision problems, denoted by $\mathcal{P}$, both the 'yes' and 'no' answers can be verified in polynomial time. Hence, $\mathcal{P}$ is a subset of $\mathcal{NP}$.

An interesting class of decision problems, called $\mathcal{NP}$*-complete* is defined using the concept of *reducibility.* A problem P_1 reduces to problem P_2 if there exists a

polynomial-time computable function g that transforms inputs for P_1 into inputs for P_2 such that x certifies that the answer is 'yes' for P_1 if and only if $g(x)$ certifies that the answer is 'yes' for P_2. The notion of reducibility is transitive: if P_1 reduces to P_2 and P_2 reduces to P_3, then P_1 reduces to P_3.

A problem is $\mathcal{NP}$-complete if the problem is in $\mathcal{NP}$ and every problem in $\mathcal{NP}$ reduces to it. Cook [33] proves that the *satisfiability* problem is $\mathcal{NP}$-complete. Garey and Johnson [41] present an extensive list of $\mathcal{NP}$-complete problems. From the transitivity of reducibility it follows that a problem in $\mathcal{NP}$ can be shown to be $\mathcal{NP}$-complete if a $\mathcal{NP}$-complete problem reduces to it.

As we have discussed above, inputs are usually encoded in binary notation. However, one may also consider a *unary* notation, wherein, e.g., the integer 5 is represented as (11111). In the *partition problem*, a set $\mathbf{A}$ of k integers a_1, ..., a_k, is given and the question is whether there exists a $\mathbf{B} \subset \mathbf{A}$ such that $\sum_{a_l \in \mathbf{B}} a_l = \sum_{a_l \in \mathbf{A}} a_l/2$. This problem is $\mathcal{NP}$-complete under a binary encoding. On the other hand, it can be solved by a dynamic programming recursion in $O(k \sum_l a_l)$ time, which is polynomial under a unary encoding. An algorithm which is polynomial under a unary coding is called a *pseudo-polynomial algorithm*. In the 3-partition problem a set $\mathbf{A}$ of $3k$ integers a_1, ..., a_{3k}, is given and the question is whether there exists a partition of $\mathbf{A}$ into k 3-element sets $\mathbf{B}_1$, ..., $\mathbf{B}_k$ such that $\sum_{a_l \in \mathbf{B}_{\hat{k}}} a_l = \sum_{a_l \in \mathbf{A}} a_l/k$ for $\hat{k} = 1, 2, ..., k$. The 3-partition problem is $\mathcal{NP}$-complete even under a unary coding and is therefore

called *strongly $\mathcal{NP}$-complete.*

The terms $\mathcal{NP}$-complete and strongly $\mathcal{NP}$-complete are used for decision problems. An optimization problem is *(strongly) $\mathcal{NP}$-hard* if its decision problem is (strongly) $\mathcal{NP}$-complete.

One of our goals is to classify the problems into $\mathcal{NP}$-hard and polynomially solvable cases. $\mathcal{NP}$-hardness of a problem has been accepted as a justification for using (i) polynomial-time algorithms that produce near-optimal solutions, and (ii) exponential-time algorithms that produce exact solutions.

1.6 Hierarchy of Scheduling Objectives

There exist some important elementary reductions (Graham et al. [51]) between scheduling problems. These reductions are shown in Figure 1.2 where $\delta \rightarrow \delta'$ implies that the decision problem with objective δ reduces to the decision problem with objective δ'.

Let $\mathcal{I}_1$ be an instance of the scheduling problem with weight w_j and due date d_j of each job j.

Let $\mathcal{P}_1$ be the following decision problem: does there exist a schedule for $\mathcal{I}_1$ with $\delta \leq \mathcal{D}$?

Let $\mathcal{I}_2$ be the instance $\mathcal{I}_1$ with due date $d'_j = 0 \ \forall j$.

Let $\mathcal{I}_3$ be the instance $\mathcal{I}_1$ with due date $d'_j = d_j + \mathcal{D} \ \forall j$.

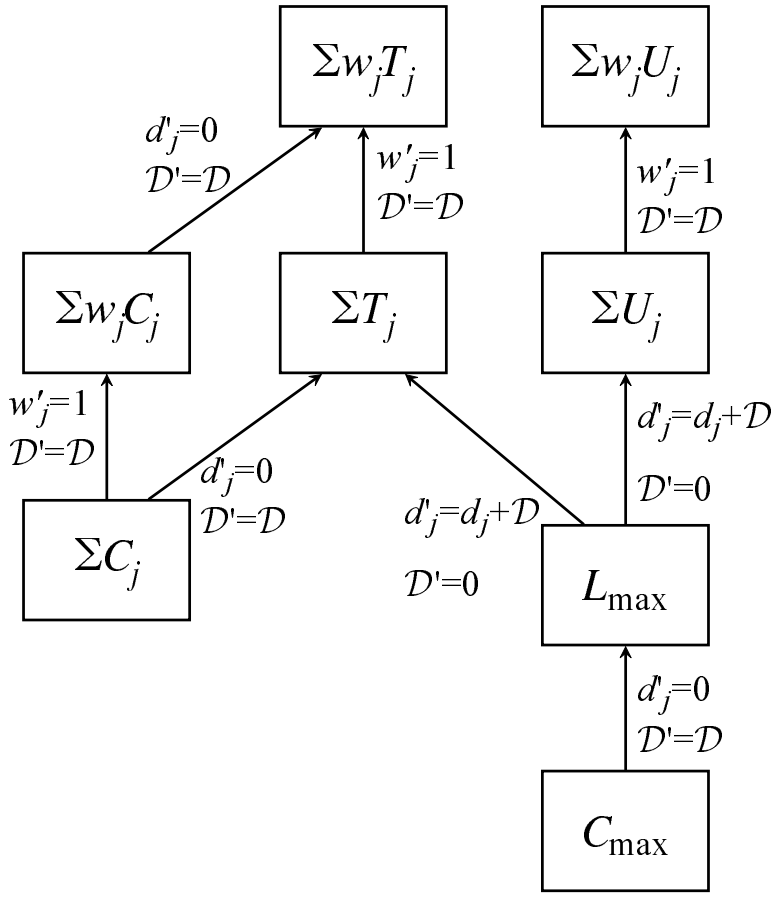

Figure 1.2: Relationship among various regular objectives

Let $\mathcal{I}_4$ be the instance $\mathcal{I}_1$ with weight $w'_j = 1\ \forall j$.

Problem $\mathcal{P}_1$ reduces to the following decision problem: does there exist a schedule for $\mathcal{I}$ with $\delta' \leq \mathcal{D}'$?, where

1. if $\delta = C_{\max}$, then $\delta' = L_{\max}$, $\mathcal{I} = \mathcal{I}_2$, $\mathcal{D}' = \mathcal{D}$;

2. if $\delta = \sum C_j$, then $\delta' = \sum w_j C_j$, $\mathcal{I} = \mathcal{I}_4$, $\mathcal{D}' = \mathcal{D}$ or $\delta' = \sum T_j$, $\mathcal{I} = \mathcal{I}_2$, $\mathcal{D}' = \mathcal{D}$;

3. if $\delta = L_{\max}$, then $\delta' = \sum T_j$, $\mathcal{I} = \mathcal{I}_3$, $\mathcal{D}' = 0$; or $\delta' = \sum U_j$, $\mathcal{I} = \mathcal{I}_3$, $\mathcal{D}' = 0$;

4. if $\delta = \sum w_j C_j$, then $\delta' = \sum w_j T_j$, $\mathcal{I} = \mathcal{I}_2$, $\mathcal{D}' = \mathcal{D}$;

5. if $\delta = \sum T_j$, then $\delta' = \sum w_j T_j$, $\mathcal{I} = \mathcal{I}_4$, $\mathcal{D}' = \mathcal{D}$; and

6. if $\delta = \sum U_j$, then $\delta' = \sum w_j U_j$, $\mathcal{I} = \mathcal{I}_4$, $\mathcal{D}' = \mathcal{D}$;

This shows the validity of all the reductions presented in Figure 1.2. Such reductions are important because if $\delta \rightarrow \delta'$ then: (i) if a problem with objective δ' can be solved in polynomial time, the problem with objective δ can also be solved in polynomial time; and (ii) if a problem with objective δ is (strongly) $\mathcal{NP}$-hard, then the problem with objective δ' is also (strongly) $\mathcal{NP}$-hard.

1.7 Preview

In Chapter 2 we present a literature survey on the theoretical development. In Chapter 3 we discuss some dominance properties for regular objectives. Dominance properties narrow down the search for an optimal schedule. In the case of two machines the dominance properties can be given a simpler interpretation. In Chapter 4 the dominance properties are used to develop a common dynamic programming approach that applies to all the fixed sequence cases except the ones with objectives $\sum U_j$ and $\sum w_j U_j$.

In Chapter 5 we revisit problems $1F2|s_i|L_{\max}$ and $1O2|s_i|L_{\max}$ and improve the algorithm's running time. We discuss a network representation which shows that

problems $1F2|s_i|L_{\max}$ and $1O2|s_i|L_{\max}$ can be interpreted as shortest path problems, if the lengths of arcs and paths are defined in a particular way.

In Chapter 6 we show that each of the problems $1F2|s_i|\sum w_jU_j$, $1O2|s_i|\sum w_jU_j$, $1F2|s_i|\sum U_j$, and $1O2|s_i|\sum U_j$ is $\mathcal{NP}$-hard but solvable in pseudo-polynomial time. We discuss two different pseudo-polynomial time algorithms for each of the problems $1F2|s_i|\sum w_jU_j$ and $1O2|s_i|\sum w_jU_j$, and report computational experience.

In Chapter 7 we consider some restricted cases of problems $1F2|s_i|\sum w_jC_j$ and $1O2|s_i|\sum w_jC_j$. We revisit some fixed sequence cases with the objective $\sum w_jC_j$ and improve the algorithm's running time. We show that fixed sequence cases with objective $L_{\max}$ and $\sum w_jC_j$ can be given a similar network representation. The results on fixed sequence cases can be used to develop an enumeration scheme. However, there are $n!$ sequences, which makes the enumeration unattractive for large n. We then show that if all $w_j = 1$, an alternative, more attractive, enumeration scheme can be used. In Chapter 8 we consider the problem $1F2|s_i|\sum C_j$. The complexity status of this problem is open. For this reason, we discuss a heuristic and a lower bounding procedure which are used in a branch and bound scheme based on the alternate enumeration scheme suggested in Chapter 7. We report performance of various implementation of the branch and bound scheme and of an integer programming formulation.

Finally, in Chapter 9 we present conclusion and possible future extensions.

Chapter 2

Literature Survey

Deterministic scheduling is one of the classical problems of operations research. In the last five decades many researchers have shown great interest in this area. The literature is quite extensive. Some of the relevant books are: Baker [11], Błażewicz, Ecker, Pesch, Schmidt and Węglarz [18], Błażewicz, Ecker, Schmidt and Węglarz [19], Brucker [23], Chrétienne, Coffman, Lenstra and Liu [28], Coffman [29], Conway, Maxwell and Miller [32], French [38], Muth and Thompson [88], Pinedo [93] and Tanaev, Gordon and Shafransky [113]. Some of the recent survey articles are: Błażewicz [14], Błażewicz, Domschke and Pesch [16], Błażewicz, Dror and Węglarz [17], Brah, Hunsucker and Shah [22], Graham, Lawler, Lenstra and Rinnooy Kan [51], Graves [52], Lawler [72], Lawler, Lenstra and Rinnooy Kan [75], Lawler, Lenstra, Rinnooy Kan and Shmoys [76] Lenstra, Rinnooy Kan and Brucker [80] and Pinson [93]. Some of the related Ph.D. dissertations are: Rinnooy Kan [99], Lenstra [79] and Van de Velde [115].

2.1 Single Machine Scheduling

2.1.1 The Classical Single Machine Problem

Two simple single machine optimization rules (French [38]) obtained in the early days of deterministic scheduling are: (i) Jackson's Earliest Due Date (EDD) rule for minimizing maximum lateness: arrange the jobs in order of non-decreasing d_j; and (ii) Smith's Weighted Shortest Processing Time rule (WSPT) for minimizing total weighted completion time: arrange the jobs in order of non-increasing w_j/t_j. These rules aroused the interest of many researchers and led them to study some other closely related problems.

Another problem solvable by a simple algorithm is $1||\sum U_j$. Moore and Hodgson (Moore [86]) show that there exists a schedule in which all non-tardy jobs are arranged according to the EDD rule and precede all the tardy jobs. The property extends to the weighted case, $1||\sum w_j U_j$. The algorithm for the problem $1||\sum U_j$ is as follows: add the jobs to the set of on-time jobs in order of nondecreasing due dates, and if the addition of job j results in job j being tardy, remove the scheduled job with the largest processing time.

If the weights are unequal, the problem $1||\sum w_j U_j$ is $\mathcal{NP}$-hard (Karp [66]). However, problem $1||\sum w_j U_j$ is pseudo-polynomially solvable. Lawler and Moore [77] give an $O(n\sum t_j)$ time dynamic programming recursion, where t_j is the processing time of job j, and Sahni [104] gives an $O(n\sum w_j)$ time recursion.

2.1.2 Single Machine Scheduling with Setup

There has been a lot of research work in the last 15 years in the area of single machine scheduling problem with setups. Allahverdi et al. [5], Potts and Kovalyov [95], Potts and Van Wassenhove [96] and Webster and Baker [121] present comprehensive surveys on the area. Some Ph.D. dissertations on scheduling problems with setups are: Julien [63], Landy [70], Sahney [101] and Santos [106].

The class of single machine family scheduling problems is a natural consequence of setups. If operations require setups, it is natural to assume that there are F families of operations, so that each operation belonging to a particular family requires a particular setup. If an operation belongs to a previously-processed operation, no new setup is needed. However, a setup time $s_{ii'}$ is required each time the machine switches from processing operations in family i to processing operations in family $i' \neq i$. If the setup times are sequence-independent, $s_{ii'} = s_{i'}$. Otherwise, setup times are sequence-dependent. For job j, processing of the operation in family i requires time t_{ij}.

Three cases of the family scheduling problem are closely related to the one-operator problem: (i) *single-operation batch availability*: each product has only a single operation and the products are delivered after the batch of operations is complete; (ii) *single-operation item availability*: each product has only a single operation and the products are delivered as soon as the operation is performed; and (iii) *multi-*

operation: each product comprises operations from one or more families.

In this section, we shall discuss single-operation cases. The multi-operation cases are discussed in the next section on one-operator scheduling problems.

Single-operation Batch Availability

A distinguishing feature of this problem is that jobs are processed in batches, and the completion time of a job is equal to the completion time of its batch. All the jobs in a batch belong to the same family. This problem is interesting even with a single family. If there is a single family, a setup time s_1 is needed between two successive batches. The problem is to find a sequence of operations, the number of batches and the batch sizes. This single-family batch availability problem will be denoted by $1|s_1, F = 1|\delta$.

Several authors consider the problem $1|s_1, F = 1|\sum C_j$ with identical processing times. Santos [105] suggest a dynamic programming approach to find the batch sizes for a given number of batches. Dobson, Karmakar and Rummel [35] and Santos and Magazine [107] give a closed form solution for the optimal number of batches and optimal batch sizes. Specifically, the optimal number of batches is $k^* = \left\lfloor \sqrt{1/4 + 2nt/s_1} - 1/2 \right\rfloor$ and the optimal batch sizes are $n/k^* + s_1(k^* + 1)/2t - ks_1/t$ for $k = 1, ..., k^*$. Here, t is the processing time of each job and $\lfloor x \rfloor$ denotes the largest integer less than or equal to a quantity x. Naddef and Santos [89] present an algorithm for the problem that runs in $O(nt/s_1)$ time.

The problem $1|s_1, F = 1|\sum C_j$ with identical processing times has been generalized in two directions. First, Coffman et al. [30] address the problem $1O2|s_1 = s_2, t_{ij} = t_i|\sum C_j$. They show in [30] that the problem can be solved with an $O(\sqrt{n}t_1t_2/s_1)$ time algorithm.

Second, Coffman et al. [31] address the problem $1|s_1, F = 1|\sum C_j$ with multiple products. They show in [31] that the problem has an optimal schedule in which the jobs are sequenced in the shortest processing time order. Coffman et al. [31] use this result to give a dynamic programming recursion for the problem, and develop an implementation which takes $O(n)$ time after the jobs are sorted. Furthermore, Coffman et al. [31] show that the problem of scheduling products with common and unique components, as introduced by Baker [12], can be modelled as the problem $1|s_1, F = 1|\sum C_j$. (Recall that Baker's problem is equivalent to $1F2|s_i, s_2 = 0, apt|\sum C_j$.) Therefore, it follows from the result of Coffman et al. [31] that the problem $1F2|s_i, s_2 = 0, apt|\sum C_j$ can be modelled as a problem $1|s_1, F = 1|\sum C_j$ and solved in $O(n)$ time after job sorting. Further relationships between $1F2|s_i|\delta$ and $1|s_1, F = 1|\delta$ are discussed in Section 3.5.

If the jobs are not identical and the processing time of every job is a part of the input, then the length of the input is $n \log t + \log s_1$. In such a case a running time of $O(n^k)$ is polynomial. For example, the algorithm of Coffman et al. [31] is polynomial. However, Shallcross [108] points out that none of the above algorithms is polynomial

for the problem $1|s_1, F = 1|\sum C_j$ with identical processing time, because the input for this problem is only n, t and s_1, hence, has the length $\log n + \log t + \log s_1$. The running time of the above algorithm is not a polynomial function of $\log n + \log t + \log s_1$. Shallcross [108] presents an alternative $O(\log t \log(nt))$ time algorithm for the problem $1|s_1, F = 1|\sum C_j$ with identical processing times.

Albers and Brucker [4] further generalize the problem of Coffman et al. [31] to the case with arbitrary processing times and arbitrary weights. Albers and Brucker show in [4] that the problem $1|s_1, F = 1|\sum w_j C_j$ can be solved in $O(n)$ time if the sequence is known and that if the sequence is unknown, the problem is strongly $\mathcal{NP}$-hard. Hochbaum and Landy [59] consider the problem $1|s_1, F = 1|\sum w_j C_j$ with identical processing times and two weights, and present an $O(\sqrt{n}\log n)$ time algorithm to solve it.

Webster and Baker [121] show that for the problem $1|s_1, F = 1|L_{\max}$ there exists an optimal schedule in which the jobs are sequenced in the Earliest Due Date (EDD) order. Webster and Baker use this fact to give an $O(n^2)$ time dynamic programming recursion for the problem. Hochbaum and Landy [58] show that for the decision version of the problem $1|s_1, F = 1|L_{\max}$, deciding whether there exists a feasible schedule with $L_{\max} \leq l$ can be done in $O(n)$ time.

For the problem $1|s_1, F = 1|\sum w_j U_j$, Hochbaum and Landy [58] show that there exists an optimal schedule in which all non-tardy jobs are sequenced in the Earliest

Due Date (EDD) order and precede all the tardy jobs. They use this result to give an $O(n^2 \min\{d_{\max}, \sum t_j + ns_1)$ time dynamic programming recursion for the weighted case and another $O(n^4)$ time dynamic programming recursion for the unweighted case.

For the problem $1|s_1, F = 1|\sum w_j U_j$, Brucker and Kovalyov [24] give an $O(n^2 \sum w_j)$ time dynamic programming recursion. Note that for unweighted case, $\sum w_j = n$. This means that for the unweighted case the algorithm of Brucker and Kovalyov [24] runs in time $O(n^3)$, which is an improvement over the running time of the algorithm given by Hochbaum and Landy [58].

Single-Operation Item Availability

The single machine item availability problems are denoted by $1|s_{ii'}|\delta$. The sequence-dependent problem is strongly $\mathcal{NP}$-hard for $\delta \in \{C_{\max}, L_{\max}\}$ (Bruno and Downey [25]) and $\delta = \sum C_j$ (Rinnooy Kan [99, see p.85]).

The one-operator two-machine problem addressed by Sahney ([101]-[103]) can be viewed (Ghosh [47], Gupta [53, 54]) as a single machine family scheduling problem with two families. Sahney [102] considers the problem $1|s_{ii'}, F = 2|\sum C_j$, develops some dominance properties and proposes a branch and bound procedure. Sahney [102] shows that there exists an optimal schedule in which jobs within a family are processed in the shortest processing time order.

Psaraftis [97] considers the problem $1|s_{ii'}, t_{ij} = t_i|\sum C_j$ in which jobs within each

family are identical, and presents a dynamic programming recursion to solve it. The approach, as he states, applies to the other objectives makespan and total weighted completion time. Let $\tilde{n} = \max\{n_i\} + 1$, where n_i is the number of jobs in family i. The dynamic programming recursion of Psaraftis [97] runs in time $O(F^2\tilde{n}^F)$. Since the algorithm is exponential in F, the practical performance may be satisfactory, only if F is small. However, as Psaraftis [97] points out, the algorithm's performance is an improvement over that of the classical dynamic programming algorithm of Held and Karp [57], which requires $O(\hat{n}^2 2^{\hat{n}})$ time, where $\hat{n} = \sum n_i$.

Dobson et al. [35] consider the sequence independent problem $1|s_i, t_{ij} = t_i|\sum C_j$ in which jobs within each family are identical. They show that there exists an optimal schedule in which all the jobs belonging to the same family are processed in a single batch and the batches are processed in non-decreasing order of $(s_i + t_i n_i)/n_i$, where t_i is the processing time of any job that belongs to family i. Hence, the problem is solvable in time $O(F \log F)$.

Monma and Potts [85] show that for the problem $1|s_{ii'}|\sum w_j C_j$, there exists an optimal schedule in which jobs are arranged in order of non-increasing w_j/t_{ij} within each family i. Monma and Potts [85] present a dynamic program which solves the problem $1|s_{ii'}|\sum w_j C_j$ in $O(F^2\tilde{n}^{F^2+2F})$ time and solves the sequence-independent case $1|s_i|\sum w_j C_j$ in $O(F^2 N^{2F})$ time.

Ahn and Hyun [3] and Gupta [54] consider the problem $1|s_{ii'}|\sum C_j$. Ahn and

Hyun [3] show that the algorithm of Psaraftis [97], originally proposed for the cases with identical jobs within a family extends to the problem $1|s_{ii'}|\sum C_j$ without any restriction on processing times. Their algorithm maintains the $O(F^2\tilde{n}^F)$ running time of the algorithm of Psaraftis [97]. As the running time of the algorithm is exponential in number of families, Ahn and Hyun [3] and Gupta [54] propose heuristic methods that may be useful if there are several families.

Gupta [53] and Potts [94] consider the problem $1|s_i, F = 2|\sum C_j$. Potts [94] uses the algorithm of Ahn and Huyn [3] and shows that the problem $1|s_i, F = 2|\sum C_j$ is solvable in $O(n^2)$ time. Potts [94] also considers a weighted case, the problem $1|s_i, F = 2|\sum w_jC_j$, and shows that a variant of the algorithm of Monma and Potts [85] solves this problem in $O(n^3)$ time.

Ghosh [47] generalizes the algorithm of Ahn and Hyun [3] to the weighted case without increasing the running time. Ghosh [47] shows that the problem $1|s_{ii'}|\sum w_jC_j$ is solvable in $O(F^2\tilde{n}^F)$ time. Ghosh also considers the problem $1|s_{ii'}, t_{ij} = t_i, w_{ij} = w_i|\sum w_jC_j$ with identical processing times and weights within a family. (Recall that Dobson et al. [35] show that for the problem $1|s_i, t_{ij} = t_i|\sum C_j$, there exists an optimal schedule in which all the jobs belonging to the same family are processed in a single batch.) Ghosh [47] states that if the setup times satisfy the triangle inequality condition, $s_{i_1i_2} + s_{i_2i_3} \geq s_{i_1i_3}$, then the above property of the problem $1|s_i, t_{ij} = t_i|\sum C_j$ extends to the problem $1|s_{ii'}, t_{ij} = t_i, w_{ij} = w_i|\sum w_jC_j$. Thus,

each family can be viewed as a single composite job. An immediate consequence of this is that the running time of the algorithm of Psaraftis [97] and Ahn and Hyun [3] reduces to $O(F^2 2^F)$ time for the problem $1|s_{ii'}, t_{ij} = t_i, w_{ij} = w_i| \sum w_j C_j$, if the setup times satisfy triangle inequality conditions. Ghosh [47] mentions that the problem $1|s_{ii'}, t_{ij} = t_i, w_{ij} = w_i| \sum w_j C_j$ is strongly $\mathcal{NP}$-hard.

Monma and Potts [85] show that for the problem $1|s_{ii'}|L_{\max}$, there exists an optimal schedule in which jobs are arranged in the Earliest Due Date (EDD) order within each family. They present a dynamic program which solves the problem $1|s_{ii'}| \sum L_{\max}$ in $O(F^2 \tilde{n}^{F^2+2F})$ time and solves the sequence-independent case $1|s_i| \sum L_{\max}$ in $O(F^2 N^{2F})$ time. Ghosh and Gupta [48] show that the dynamic programming approach of Psaraftis [97], Ahn and Hyun [3] and Ghosh [47] for various cases of the problem $1|s_{ii'}| \sum (w_j) C_j$ applies to the problem $1|s_{ii'}|L_{\max}$ without any change in the running time. Thus, Ghosh and Gupta [48] improve the running time of the algorithm for the problem $1|s_{ii'}|L_{\max}$ to $O(F^2 \tilde{n}^F)$ time. Bruno and Downey [25] show that the problem $1|s_i|L_{\max}$ is $\mathcal{NP}$-hard.

Monma and Potts [85] show that for the problem $1|s_{ii'}| \sum w_j U_j$, there exists an optimal schedule in which the early jobs are arranged in the Earliest Due Date (EDD) order within each family. They present a dynamic programming approach which solves the problem $1|s_{ii'}| \sum U_j$ in $O(\tilde{n}^{F+1})$ time and problem $1|s_{ii'}| \sum w_j U_j$ in $O(\tilde{n}^F \sum w_j)$ time. Since problem $1|s_i|L_{\max}$ is $\mathcal{NP}$-hard (Bruno and Downey [25]),

problem $1|s_{ii'}|\sum U_j$ is also $\mathcal{NP}$-hard.

2.2 One-Operator Scheduling Problems

Baker [12], Coffman et al. [30], Julien [63] and Santos [106, Chapter 4] have pioneered developments on the problem of scheduling products that require multiple setups on a single facility. Santos [106] considers the problem $1Fm|s_i, t_{ij} = t_i|\sum C_j$ with a batch availability assumption. An important dominance property he observes is that a machine i is never set up as long as there is a job processed on machine i but not on all the machines. Coffman et al. [30] consider the problem $1O2|s_1 = s_2$, $t_{ij} = t_i|\sum C_j$. They show that the search for an optimal schedule can be narrowed down to the schedules in which the operator switches from machine i to i' only if the number of jobs processed on machine i is strictly more than the number of jobs processed on machine i'. Julien [63] considers the problem $1Om|s_i, n_{ij} \geq 0|\sum C_j$. He observes that for this problem there exists an optimal schedule in which (i) job-orders are same on all machines; (ii) a machine i is never set up as long as there is a job processed on machine i but not on all the machines; and (iii) if $n_{ij} > 0$, then a switch from machine i to i' takes place only after processing at least one job on machine i which is not processed on all the machines. He states that the properties extend to any regular objective.

A huge literature exists on the one-operator, two-machine problem with $\sum C_j$

objective. Baker [12] points out that Santos and Magazine [107] and Dobson et al. [35] consider the problem $1|s_1, F = 1|\delta$ with identical jobs without considering the fact that the demand, n of a component is derived from the demand of the finished products which are assembled from more than one component. To overcome this limitation, Baker [12] formulates a problem of scheduling products with common and unique components with the assumption of batch availability for both common and unique components. This problem is equivalent to the problem $1F2|s_i, apt|\sum C_j$. Baker [12] shows that there exists an optimal schedule in which jobs are processed in the shortest processing time order. Baker points to a resemblance between the problem and the dynamic lot-sizing problem of Wagner and Whitin [118, 119]. An immediate consequence of this observation is the existence of an $O(n^2)$ time algorithm for the problem $1F2|s_i, apt|\sum C_j$.

Baker's problem focussed the attention of many researchers:

- Coffman et al. [31] show that Baker's problem reduces to the single machine problem $1|s_1, F = 1|\sum C_j$ with batch availability. By so doing they improve the running time to $O(n)$ after job sorting.

- Aneja and Singh [7] point out that a problem with more than one common component cannot be treated as a problem with a single common component, as was suggested by Baker. However, they show that the problem with m' common components can be solved by solving m' problems of the type $1F2|s_i, apt|\sum C_j$.

- Vickson et al. [117] and Sung and Park [111] consider the problem with item availability assumption. Thus, Vickson et al. [117] and Sung and Park [111] consider the problem $1O2|s_i, apt|\sum C_j$. Vickson et al. [117] present a dynamic programming recursion and Sung and Park [111] present a dynamic program and a branch and bound algorithm. Interestingly, Sung and Park [111] observe that the branch and bound algorithm performs better than the dynamic program, although each has $O(n^2)$ running time. Gerodimos et al. [45] give another $O(n^2)$ time algorithm for the problem $1O2|s_i, apt|\sum C_j$. The algorithm of Julien and Magazine [64], although presented in a different context, also solves the problem $1O2|s_i, apt|\sum C_j$ in $O(n^2)$ time.

- Ding [34] relaxes the assumption of agreeable processing times and develops some schedule improvement rules and a heuristic method.

- Rana and Singh [98] consider the problem with the multiple objectives of total completion time, makespan and another objective defined in their paper.

Gerodimos et al. [44, 45, 46] consider various cases of problems $1Om|s_i, n_{ij} \geq 0|L_{\max}$, $1Om|s_i, n_{ij} \geq 0|\sum C_j$ and $1Om|s_i, n_{ij} \geq 0|\sum w_j U_j$. The problem $1Om|s_i, n_{ij} \geq 0|L_{\max}$ is $\mathcal{NP}$-hard but is solvable in $O(m^2 n^m)$ time using the algorithm of Ghosh and Gupta [48] for the problem $1|s_{ii'}|L_{\max}$. A similar dynamic program solves the problem $1Om|s_i, n_{ij} \geq 0, apt|\sum C_j$ in $O(n^m)$ time. The special case of two machines

can be solved in $O(n^2)$ time if the objective is $L_{\max}$ or if the processing times are agreeable and the objective is $\sum C_j$.

The problem $1Om|s_i, n_{ij} \geq 0|\sum w_j U_j$ is $\mathcal{NP}$-hard even when all $w_j = 1$ but is solvable in $O(nd_{\max}^m)$ time. The special case of two machines, identical processing time for the common components and $w_j = 1$ is polynomially solvable.

2.3 Shop Problems

2.3.1 Flow Shop

One of the first developments in deterministic scheduling is Johnson's algorithm (French [38]), which solves the problem $F2|\ |C_{\max}$ with a simple rule: first arrange the jobs with $t_{1j} \leq t_2$ in order of non-decreasing t_{1j}, and then arrange the remaining jobs in order of non-increasing t_{2j}. Conway et al. [32] observe that for $Fm|\ |C_{\max}$ there exists an optimal schedule with the same processing order on machines 1 and 2 and the same processing order on machines m and $(m-1)$. This implies that for $F3|\ |C_{\max}$ there exists an optimal schedule in which all the jobs are processed in the same order on all machines. Still, the problem $F3|\ |C_{\max}$ is strongly $\mathcal{NP}$-hard (Garey et al. [42]). A number of two-machine flow shop problems are also strongly $\mathcal{NP}$-hard. These include $F2|\ |L_{\max}$ (Lenstra et al. [80]) and $F2|\ |\sum C_j$ (Garey et al. [42]).

2.3.2 Open Shop

Problem $O2|\ |C_{\max}$ admits a polynomial-time algorithm (Gonzalez and Sahni [50]). The problem is solvable by the longest alternate processing time (LAPT) rule: whenever a machine is freed select the job waiting for processing with the longest processing time on the other machine (Pinedo [93]). However, the problem $O3|\ |C_{\max}$ is $\mathcal{NP}$-hard (Gonzalez and Sahni [50]) and a number of open shop problems are strongly $\mathcal{NP}$-hard. These include $O2|\ |L_{\max}$ (Lawler et al. [73, 74]) and $O2|\ |\sum C_j$ (Achugbue and Chin [1]).

2.4 Resource Constrained Scheduling

In a resource constrained scheduling problem, an operation may require some additional resources besides a machine. All resources required by the operation are allocated to it all the time during its execution. At no time may total resource requirements exceed resource availabilities.

Błażewicz et al. [21] classify the resource constrained problems using the $\alpha|\beta|\gamma$ notation. The resource constrained environment is specified by an entry $res\lambda_1\lambda_2\lambda_3$ in the β field. If λ_1 is a positive integer, then the number of additional resources is λ_1. For example, $\lambda_1 = 2$ if workers and tools are additional resources. If $\lambda_1 =$ ".", the number of additional resources is unspecified and is a part of the input. If λ_2 is a positive integer, then the total amount of each resource available at any given time

is a constant and equal to λ_2. For example, $\lambda_2 = 3$ if 3 workers and 3 pieces of tools are available in addition to machines at any given time. If $\lambda_2 =$ ".", the resource availability is unspecified and is a part of the input. If λ_3 is a positive integer, then each operation requires at most λ_3 unit of any resource at any given time. If $\lambda_3 =$ ".", the maximum resource requirement is unspecified and is a part of the input.

The closest notation for $1Om|s_i|\gamma$ and $1Fm|s_i|\gamma$ is then $Om|res111, s_i|\gamma$ and $Fm|res111, s_i|\gamma$ respectively. In each

case, the one-operator scheduling problem has an additional piece of information that each operation requires at least one unit of additional resource, i.e., the operator. Results for some flow shop and open shop problems with one additional resource are shown in Table 2.1. For a comprehensive survey on resource constrained scheduling, we refer the reader to Błażewicz, Cellary, Słowiński and Węglarz [15] and Błażewicz, Lenstra and Rinnooy Kan [21] and the recent books Błażewicz, Ecker, Pesch, Schmidt and Węglarz [18] and Błażewicz, Ecker, Schmidt and Węglarz [19].

2.5 Reentrant Flow Shop

In the production of Very Large Scale Integrated (VLSI) circuits or wafer fabrication there may be 250 or more different stages and 100 workstations (Lane and Sidney [71]). Certain workstations, called hubs (Lane and Sidney [71], Kubiak et al. [69]) may be revisited at a number of stages. Kubiak et al. [69] present two other examples

Problem	Complexity	Reference
$F2\|res111\|C_{\max}$	Strongly $\mathcal{NP}$-hard	Błażewicz et al. [21]
$F2\|res111, t_{ij}=1\|C_{\max}$	$O(n)$	Błażewicz et al. [21]
$F2\|res1.., t_{ij}=1\|C_{\max}$	$O(n \log n)$	Röck [100]
$F3\|res111, t_{ij}=1\|C_{\max}$	$O(n)$	Süral et al. [112]
$F3\|res1.., t_{ij}=1\|C_{\max}$	Strongly $\mathcal{NP}$-hard	Röck [100]
$Fm\|res1.1, t_{ij}=1\|C_{\max}$	$O(n^{2^{m+1}})$	Błażewicz et al. [20]
$O2\|res111\|C_{\max}$	$O(n)$	Kubiak [68]†
$O2\|res1..\|C_{\max}$	$O(n^3)$	Jurisch and Kubiak [65]
$O3\|res1.., t_{ij}=1\|C_{\max}$	Strongly $\mathcal{NP}$-hard	Błażewicz et al. [15]

†Ph.D. thesis of Kubiak is in Polish. Jurisch and Kubiak [65] cite the result on $O2|res111|C_{\max}$.

Table 2.1: Some results on resource constrained scheduling

of such revisits. First, during a Printed Circuit Board (PCB) assembly the same machine installs surface-mounted devices to the PCB upperside and lowerside at two different stages. Second, in a painting shop parts have to visit the painting and baking divisions alternately for different coats of paint. Lev and Adiri [81] imply that during replacement of a damaged internal part of a complex piece of equipment, the same resource may be required once for removing a part and again for restoring the part. Morton and Pentico [87] present a case study on a nuclear fuel tube shop which produces zircalloy tubes of several diameters, sizes, and types. Each tube goes through a single annealing furnace at three different stages.

In each of the above examples jobs visit a machine more than once. Although none of these examples include setup times, theoretical developments on these problems shed some light on the extension of one-operator problem to multi-operator cases

which is observed if machines are arranged in a U-turn layout and operators are assigned to machines both in the beginning of the line and in the end of the line (Miltenburg [84], Sparling [109], Urban [114]).

Consider a production system with two operators and three machines. Suppose that every job requires exactly one operation on each machine, and the jobs are processed first on machine 1, then on machine 2, and finally on machine 3. If one operator is assigned to machines 1 and 3 and the other operator is assigned to machine 2, the problem is $\mathcal{NP}$-hard for the makespan objective (Lane and Sidney [71], Wang et al. [120], Lev and Adiri [81]). A special case of the problem with $t_{1j} = 0$ is equivalent to the two-machine flow shop problem, which is strongly $\mathcal{NP}$-hard for total completion time objective (Garey et al. [42]) and maximum lateness objective (Lenstra et al. [80]). This means that the two-operator, three-machine problem is strongly $\mathcal{NP}$-hard for total completion time objective and maximum lateness objective.

If one operator is assigned to machines 1 and 2 and the other operator is assigned to machine 3, then for any regular objective, the problem without any setup times reduces to a two-machine flow shop problem. The reason for this is that in the two-operator, three-machine problem there exists an optimal schedule in which the operator assigned to the first two machines processes every job contiguously on both machines. However, a positive setup time on machine 1 makes the problem $\mathcal{NP}$-hard even for the makespan objective (Cheng and Wang [27]).

Chapter 3

Regular Objectives

In this chapter we shall discuss some dominance properties. Santos [106], Coffman et al. [30] and Julien [63] observe some dominance properties for problems $1Fm|s_i, t_{ij} = t_i|\sum C_j$, $1O2|s_1 = s_2, t_{ij} = t_i|\sum C_j$ and $1Om|s_i, n_{ij} \geq 0|\sum C_j$, respectively. In this chapter, we shall show that these properties extend to the cases of $1Om|s_{ii'}, n_{ij} \geq 0|\delta$ and $1Fm|s_{ii'}, n_{ij} \geq 0|\delta$. We shall discuss the case of two machines separately. In the case of two machines, the properties can be given a simpler interpretation.

In the next chapter, the dominance properties will be used to develop a common dynamic programming scheme that can be used for both the open shop and flow shop cases and makespan, maximum lateness and total (weighted) completion time objectives.

3.1 Preliminary

A schedule is a time table for performing tasks such as operations and setups, and for utilizing resources such as operators and machines. A schedule can be represented by a Gantt-chart (see, e.g., French [38]). In a Gantt-chart, each operation and setup is represented by a block on the time-axis that represents a machine or operator, the length of a block being proportional to the processing time of the operation or setup that it represents.

The k-th operation of job j on machine i is denoted by (i, j, k). If every job has exactly one operation on each machine, (i, j) denotes the operation of job j on machine i. Setup on machine i is denoted by S_i. Machine i is represented by M_i.

Example 3.1 *Consider an instance of* $1O2|s_i|\delta$ *with* $s_1 = 1$, $s_2 = 1$ *and the following processing times:*

j	1	2	3
t_{1j}	1	1	2
t_{2j}	1	2	1

Consider a schedule σ in which the operator first processes job 2 on machine 1, then job 1 on both machines, then job 3 on machine 1, and then jobs 2 and 3 on machine 2. The schedule is shown in the Gantt-chart in Figure 3.1.

An idle period of the operator is unnecessary if some operation or setup can be started earlier without altering the sequence. Such an adjustment of the start time is equivalent to moving the block that represents the operation or setup to the left in the

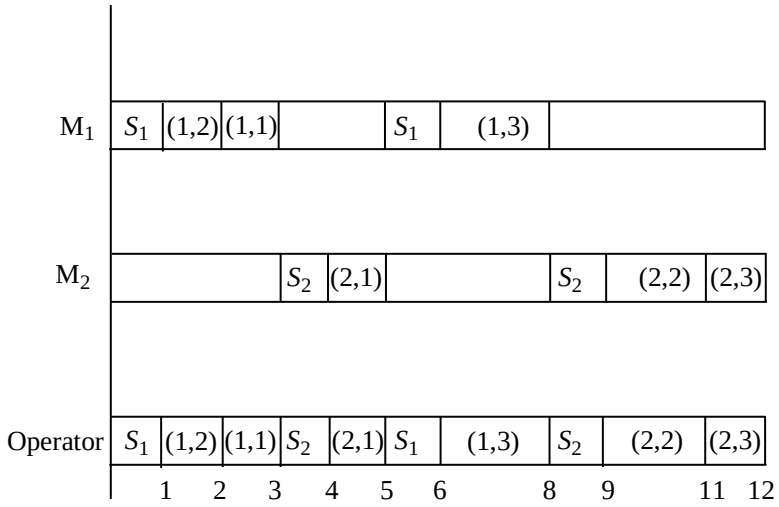

Figure 3.1: A Gantt Chart

Gantt-chart. This type of adjustment is called a *left-shift.* A schedule is called *semi-active* if no left-shift can be made. For any regular objective, the set of semi-active schedules dominates the set of all schedules; see e.g., Baker [11] or French [38]. We shall thus limit our search for an optimal schedule to the set of semi-active schedules.

Given a sequence of operations, we can generate the setup requirements and the semi-active schedule that corresponds to the sequence. Thus, we can get start and completion times of every operation and setup. Since we limit our discussion to the set of semi-active schedules, a sequence of operations is sufficient to describe a schedule. For example, the schedule shown in Figure 3.1 is $(1,2)$, $(1,1)$, $(2,1)$, $(1,3)$, $(2,2)$, $(2,3)$.

We call a machine i the *current machine* at time t if the operator completes the

setup operation of machine i at some time $t' \leq t$ and does not start a setup operation of any other machine between times t' and t. For example, between times 1 and 3, machine 1 is the current machine in the schedule shown in Figure 3.1.

The following rescheduling operations can be carried out without any additional setups: (i) interchanging two operations on the same machine. (For example, interchange jobs 1 and 2 on machine 1.) The resulting schedule is shown in Figure 3.2(a); and (ii) moving an operation to immediately before or after another operation on a same machine. (For example, move operation (1,2) to immediately before operation (1,3).) The resulting schedule is shown in Figure 3.2(b)

Moving an operation, however, may produce a schedule with some contiguous setups. For example, consider the schedule σ' obtained from σ by moving operation (1,3) to immediately after operation (1,1). Schedule σ' is shown in Figure 3.2(c). The second setup of machine 1 and the second setup of machine 2 are contiguous in schedule σ'. If two setups are contiguous, we can remove the first one or both. If the setup times are sequence independent as in Example 3.1, or if the setup times are sequence dependent and satisfy the triangle inequality condition, $s_{i_1 i_2} + s_{i_2 i_3} \geq s_{i_1 i_3}$, then such removals do not increase any completion time.

A job j is in the *inventory* of machine i at time t, if job j is processed on machine i on or before time t and has not yet left the shop floor. Thus, the definition of inventory is similar to the definition of *echelon inventory* in the multi-stage inventory

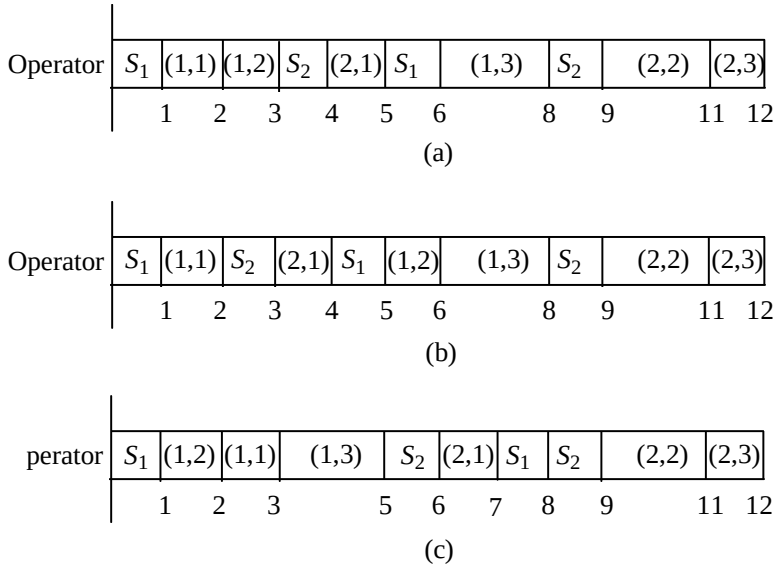

Figure 3.2: (a) Interchange (1,1) and (1,2); (b) move (1,2) before (1,3); and (c) move (1,3) before (1,1).

control analysis. For example, consider the schedule shown in Figure 3.2(a). Jobs 1 and 2 are processed on machine 1 before time 3. Until that time none of these two jobs are processed on machine 2. Hence, both jobs 1 and 2 are in the in the inventory of machine 1 at time 3. At time 5 job 1 is completed on machine 2 and leaves the shop floor. Hence, the inventory of machine 1 contains only job 2 at time 5. In this way, whenever a job is completed on a machine, it is added to the inventory of the machine, and whenever a job leaves the shop floor, it is removed from the inventory of all machines.

Consider two schedules σ and σ'. For any i and k let $\mathbf{O}$ and $\mathbf{O}'$ be the set of operations completed before the k-th setup of machine i in schedules σ and σ' respec-

tively. Let $I_{i,k}(\sigma)$ and $I_{i,k}(\sigma')$ be the number of jobs in the inventory of machine i at the time of the k-th setup of machine i in schedules σ and σ' respectively. Although inventory, I depends on time, t we do not use an argument t because t is defined by the quantities i, k and σ or σ'. Observe the following:

1. If $\mathbf{O}'= \mathbf{O}$, then $I_{i,k}(\sigma') = I_{i,k}(\sigma)$.

2. If a job j is processed on some machine $i' \neq i$, then the inventory of machine i may decrease by 1 (if job j leaves the shop floor) or remain unchanged (if job j does not leave the shop floor). Hence, if $\mathbf{O}'= \mathbf{O} \cup \{(i', j)\}$ for some $i' \neq i$ and a job j, then $I_{i,k}(\sigma) - 1 \leq I_{i,k}(\sigma') \leq I_{i,k}(\sigma)$. More precisely, if $\mathbf{O}'$ contains all operations of job j, then $I_{i,k}(\sigma') = I_{i,k}(\sigma) - 1$; otherwise $I_{i,k}(\sigma') = I_{i,k}(\sigma)$.

3. If a job j is processed on machine i, then the inventory of machine i may increase by 1 (if job j does not leave the shop floor) or remain unchanged (if job j leaves the shop floor). Hence, if $\mathbf{O}'= \mathbf{O} \cup \{(i, j)\}$ for some job j, then $I_{i,k}(\sigma) \leq I_{i,k}(\sigma') \leq I_{i,k}(\sigma) + 1$. More precisely, if $\mathbf{O}'$ contains all operations of job j, then $I_{i,k}(\sigma') = I_{i,k}(\sigma)$; otherwise $I_{i,k}(\sigma') = I_{i,k}(\sigma) + 1$.

3.2 The Case of m Machines

First, let us show that if a job has more than one operation on the same machine, it is better to not split the operations.

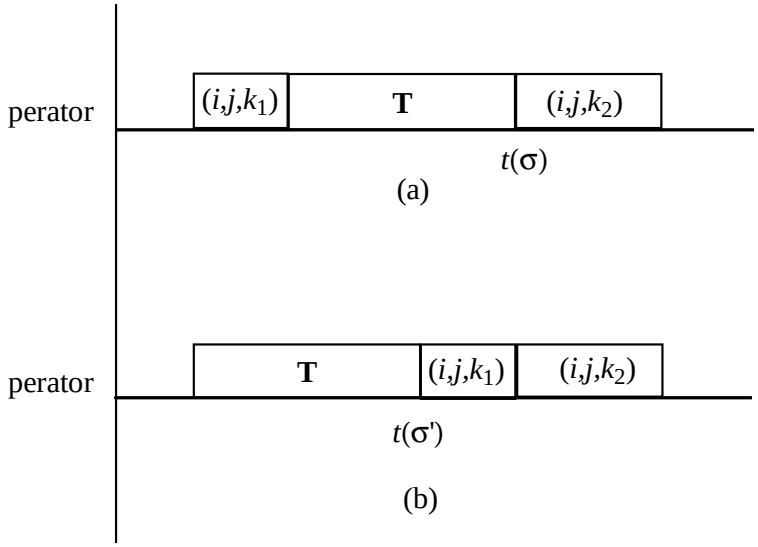

Figure 3.3: (a) Two operations of a same job on a same machine are not contiguous; and (b) a modified schedule.

Theorem 3.1 *For both problems* $1Om|s_{ii'}, n_{ij} \geq 0|\delta$ *and* $1Fm|s_{ii'}, n_{ij} \geq 0|\delta$, *there exists an optimal schedule in which all the operations of a job on the same machine are processed contiguously in one setup on that machine.*

Proof: Given any schedule that violates the condition of the Theorem, we will show that we can repeatedly apply a rescheduling procedure (see Figure 3.3) to obtain a schedule which is not worse than the original one but satisfies the condition stated in the Theorem.

To see this, consider any schedule σ which violates the result stated in the Theorem. Let $t(\sigma)$ be the maximum time when a non-empty set **T** of operations and setups is completed, such that: **T** starts immediately after operation (i, j, k_1), **T** does

not contain any operation of the type (i, j, k), and $\mathbf{T}$ is completed immediately before operation (i, j, k_2). Remove operation (i, j, k_1). Schedule every operation and setup in $\mathbf{T}$ to start t_{i,j,k_1} time units earlier. Schedule the operation (i, j, k_1) to start at time $t(\sigma) - t_{i,j,k_1}$. Call the modified schedule σ'. Note that $t(\sigma') \leq t(\sigma) - t_{i,j,k_1}$. In schedule σ', (i, j, k_1) is the only operation whose completion time is greater than in σ. However, the completion time of job j (and all other job) is no greater in σ' than in σ because completion of job j occurs after the completion of operation (i, j, k_2). Thus, the objective function value does not increase when σ is changed to σ'. Now, if σ' violates the Theorem, set $\sigma \leftarrow \sigma'$ and repeat the procedure.

Observe that the set of operations processed after $t(\sigma')$ in σ' comprises all the operations processed after $t(\sigma)$ in σ together with some other operations that include at least operation (i, j, k_1). Hence, the number of operations processed after $t(\sigma')$ in σ' is at least one more than the number of operations processed after $t(\sigma)$ in σ. This means that we need to apply the above rescheduling at most $\sum\sum n_{ij}$ times to obtain a schedule that satisfies the result stated in the Theorem. ■

An important implication of Theorem 3.1 is that the conditions $n_{ij} \geq 0$ and $n_{ij} \geq 1$ are equivalent to $0 \leq n_{ij} \leq 1$ and $n_{ij} = 1$, respectively. Henceforth, we shall consider only the cases with $0 \leq n_{ij} \leq 1$ and $n_{ij} = 1$.

In this thesis, our default assumption is that every job has exactly one operation on each machine. Hence, $n_{ij} = 1$. However, some of the results in Chapter 3 will be

developed using a relaxed assumption $0 \leq n_{ij} \leq 1$, which means that some job may not have any operation on some machine at all.

Results similar to the above have been observed by Julien and Magazine [64] and Gupta et al. [55]. Julien and Magazine consider the case of an open shop environment, sequence-independent setup times and total completion time objective. Gupta et al. consider the case of an open shop environment, sequence-independent setup times, and two objectives one of which is makespan.

Theorem 3.2 *(a) For problem* $1Om|s_{ii'}, n_{ij} \geq 0|\delta$, *there exists an optimal schedule in which all the operations of a job on machines* $\{i' : s_{ii'} = 0 \ \forall i\}$ *are processed contiguously. (b) For problem* $1Fm|s_{ii'}, n_{ij} \geq 0|\delta$, *there exists an optimal schedule in which all the operations of a job on contiguous machines* i, $i+1$, ..., i' *such that* $s_{i(i+1)} = s_{(i+1)(i+2)} = ... = s_{(i'-1)i'} = 0$, *are processed contiguously.*

Proof: Consider part (a). Let $\mathbf{M} = \{i' : s_{ii'} = 0 \ \forall i\}$. Consider any schedule σ which violates the stated result. Let $t(\sigma)$ be the maximum time when a non-empty set $\mathbf{T}$ of operations and setups is completed, such that: $\mathbf{T}$ starts immediately after operation (i_1, j), $\mathbf{T}$ is completed immediately before operation (i_2, j) and $\mathbf{T}$ does not contain any operation (i_3, j), where $i_1 \in \mathbf{M}$, $i_2 \in \mathbf{M}$, and $i_3 \in \mathbf{M}$. Remove operation (i_1, j). Schedule every operation and setup in $\mathbf{T}$ to start $t_{i_1,j}$ time units earlier. Schedule the operation (i_1, j) to start at time $t(\sigma) - t_{i_1,j}$. Call the modified schedule σ'. Observe that $t(\sigma') \leq t(\sigma) - t_{i_1,j}$. Using arguments similar to the ones used in the proof of

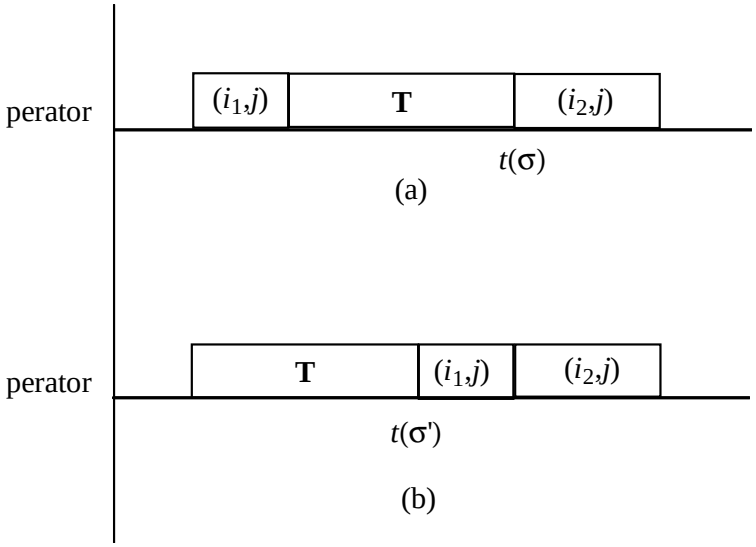

Figure 3.4: (a) Two operations of a same job on two machines with zero setup times are not contiguous; and (b) a modified schedule.

Theorem 3.1, we have $\delta(\sigma') \leq \delta(\sigma)$. If σ' does not satisfy the result stated in the Theorem, set $\sigma \leftarrow \sigma'$ and repeat the procedure.

Figure 3.4 illustrates an iteration. Using arguments similar to the ones used in the proof of Theorem 3.1 we can show that we need to apply the above rescheduling at most $\sum \sum n_{ij}$ times to obtain a schedule that satisfies the condition stated in part (a). This completes the proof of part (a). Proof of part (b) is similar. ■

For the open shop case we shall henceforth assume that there is at most one machine which does not require any setup time, or for which setup time is imbedded in the processing time. If more than one machine possesses such a property, we can get an equivalent problem by replacing all such machines with a single machine. More

precisely, if $|\mathbf{M}| > 1$, where $\mathbf{M} = \{i' : s_{ii'} = 0 \ \forall i\}$, we can get an equivalent problem in which a new machine, i (with $s_{i''i} = 0 \ \forall i''$ and $t_{ij} = \sum_{i' \in \mathbf{M}} t_{i'j}$) replaces all the machines $i' \in \mathbf{M}$. Similarly, for the flow shop case we shall assume that for three machines i, $(i+1)$, $(i+2)$, $s_{i(i+1)} = 0 \Rightarrow s_{(i+1)(i+2)} > 0$ and $s_{(i+1)(i+2)} = 0 \Rightarrow s_{i(i+1)} > 0$.

Intuitively, it is better to maintain the same job-order on all machines. In the following, we shall prove that the statement is true for any regular objective. A similar result has been obtained by Santos [106]. Note that for some non-regular objectives, it may be better to have different job-orders on different machines; see Gupta et al. [55] and Rana and Singh [98].

Theorem 3.3 *For both problems* $1Om|s_{ii'}, n_{ij} \geq 0|\delta$ *and* $1Fm|s_{ii'}, n_{ij} \geq 0|\delta$, *there exists an optimal schedule in which job-orders are the same on all machines.*

Proof: Consider any schedule σ which violates the stated result. By relabelling, if necessary, we may assume that if $j < j'$, then the last operation of job j is scheduled before the last operation of job j'. We shall repeatedly apply the following rescheduling procedure (see Figure 3.5) to obtain a schedule which is not worse than the original schedule but satisfies the result stated in the Theorem. Throughout, we maintain the property that if $j < j'$, then job j is completed before job j'.

Let $t(\sigma)$ be the maximum time when a set $\mathbf{T}$ of operations and setups is completed, such that: $\mathbf{T}$ starts immediately after operation (i, j_1) and $\mathbf{T}$ is completed

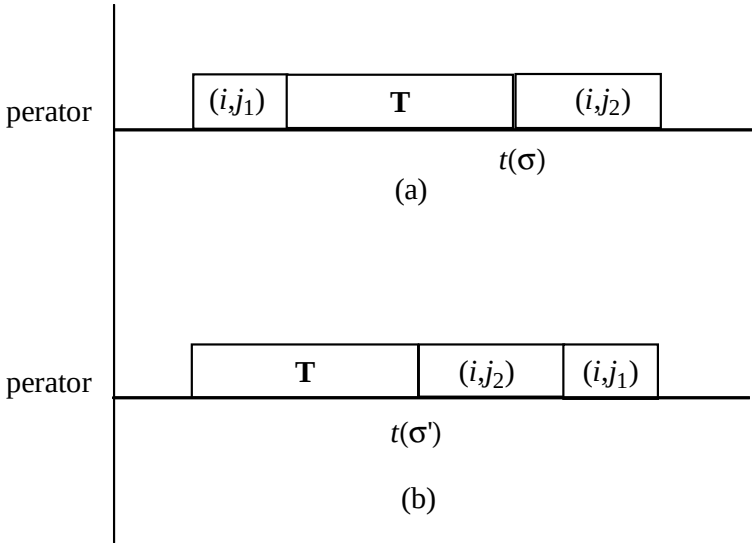

Figure 3.5: (a) Two jobs on the same machine are not in the order of their completion; and (b) a modified schedule.

immediately before operation (i, j_2), for some $j_1 > j_2$. For any job j, let C_j and $C_{i,j}$ be the completion times of job j and operation (i, j), respectively, in schedule σ. Remove operation (i, j_1). Schedule every operation and setup in $\mathbf{T} \cup \{(i, j_2)\}$ to start t_{i,j_1} time units earlier. Schedule the operation (i, j_1) to start at time $t(\sigma) + t_{i,j_2} - t_{i,j_1}$. Call the modified schedule σ'. Let C'_j be the completion time of job j in schedule σ'. Observe that $t(\sigma') \leq t(\sigma) - t_{i,j_1}$. Using arguments similar to the ones used in the proof of Theorem 3.1, we have $\delta(\sigma') \leq \delta(\sigma)$. If σ' does not satisfy the result stated in the Theorem, set $\sigma \leftarrow \sigma'$ and repeat the procedure.

Define $\mathbf{J}_1 = \{\hat{\jmath} :$ last operation of job $\hat{\jmath}$ precedes (i, j_1) in schedule $\sigma\}$, $\mathbf{J}_2 = \{\hat{\jmath} :$ last operation job $\hat{\jmath}$ is in $\mathbf{T} \cup \{(i, j_2)\}\}$ and $\mathbf{J}_3 = \{\hat{\jmath} :$ last operation of job $\hat{\jmath}$ follows

(i, j_2) in schedule $\sigma\}$. Since $j_1 > j_2$, there must be at least one operation of j_1 that follows (i, j_2) in schedule σ. Hence, $j_1 \in \mathbf{J}_3$. The relative order in which jobs in $\mathbf{J}_l$ are completed is the same in both σ and σ' for each $l = 1, 2, 3$. Job $j \in \mathbf{J}_l$ is completed before job $j' \in \mathbf{J}_{l+1}$ in both σ and σ' for each $l = 1, 2$. Hence, the order in which jobs are completed is the same in both σ and σ'. Hence, if $j < j' \Rightarrow C_j \leq C_{j'}$, then $j < j' \Rightarrow C'_j \leq C'_{j'}$.

Using arguments similar to the ones used in the proof of Theorem 3.1 we can show that we need to apply the above rescheduling at most $\sum \sum n_{ij}$ times to obtain a schedule that satisfies the result stated in the Theorem. ■

In this thesis we assume that jobs are available at time zero and remain available during the entire scheduling period. We shall now develop a result which shows that it is better to avoid setting up a machine with positive inventory. Similar results are obtained by Santos [106] and Julien [63].

Theorem 3.4 *For both problems* $1Om|s_{ii'}, n_{ij} \geq 0|\delta$ *and* $1Fm|s_i|\delta$ *there exist optimal schedules in which: (i) job-orders are same on all machines; and (ii) a machine with positive inventory is not set up.*

Proof: It follows from Theorem 3.3 that there exists an optimal schedule which satisfies result (i) stated in the Theorem. Consider any schedule σ which satisfies result (i) but not result (ii). Let $t(\sigma)$ be the maximum time when a machine with positive inventory is set up. Then, there exist i, j and k, such that: the k-th setup

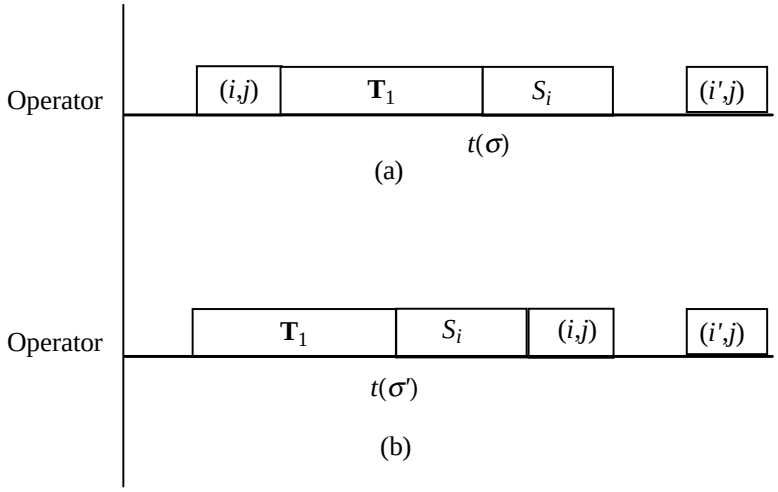

Figure 3.6: (a) A machine with positive inventory is set up; and (b) modification in case 1.

of machine i starts at time $t(\sigma)$, job j is the last processed job on machine i before the k-th setup of machine i, and job j is completed on all machines some time after the k-th setup of machine i. Let s be the time required by the k-th setup of machine i. The completion time C_j of job j satisfies $C_j \geq t(\sigma) + s$.

Case 1: Suppose that (i) the machine environment is a flow shop and operation $(i+1, j)$ is processed after $t(\sigma)$; or (ii) the machine environment is an open shop. Let $\mathbf{T}_1$ be the set of operations and setups that is processed after operation (i, j) and before the k-th setup of machine i. Remove operation (i, j). Schedule every operation and setup in $\mathbf{T}_1$ and the k-th setup of machine i to start $t_{i,j}$ time units earlier. Schedule the operation (i, j) to start at time $t(\sigma) + s - t_{i,j}$. Figure 3.6 illustrates the case.

Case 2: Suppose that the machine environment is a flow shop and operation $(i+1, j)$ is processed before $t(\sigma)$. Operation $(i, j+1)$ is processed immediately after the k-th setup of machine i. Operation $(i, j+1) \in \mathbf{O}$, where batch $\mathbf{O}$ of operations immediately follows the k-th setup of machine i. Let $\mathbf{T}_2$ be the set of operations and setups that is processed after $\mathbf{O}$ and before operation $(i+1, j+1)$. Set $\mathbf{T}_2$ is non-empty because it follows from the choice of $t(\sigma)$ that job j is completed before $(i+1, j+1)$. Furthermore, set $\mathbf{T}_2$ starts with a setup. Let t be the time required to process all operations and setups in $\mathbf{T}_2$. Remove all operations and setups in $\mathbf{T}_2$. Schedule every operation and setup in $\mathbf{O}$ and the k-th setup of machine i to start t time units later. Schedule the first setup in $\mathbf{T}_2$ at time $t(\sigma)$ and process all the operations and setups in $\mathbf{T}_2$ in the sequence in which they appear in σ. Figure 3.7 illustrates the case.

Call the modified schedule σ'. Using arguments similar to the ones used in the proof of Theorem 3.1, we have $\delta(\sigma') \leq \delta(\sigma)$. If σ' does not satisfy the result stated in the Theorem, set $\sigma \leftarrow \sigma'$ and repeat the procedure.

The order in which jobs are completed is the same in both σ and σ'. Sets $\mathbf{T}_1$ and $\mathbf{T}_2$ do not contain any operation on machine i. Hence, the job-order on machine i is the same in both σ and σ'. On any other machine, $i' \neq i$, the relative order of operations and setups is the same in both σ and σ'. This means that as the job-orders on all machines are same in σ, job-orders on all machines are same in σ'.

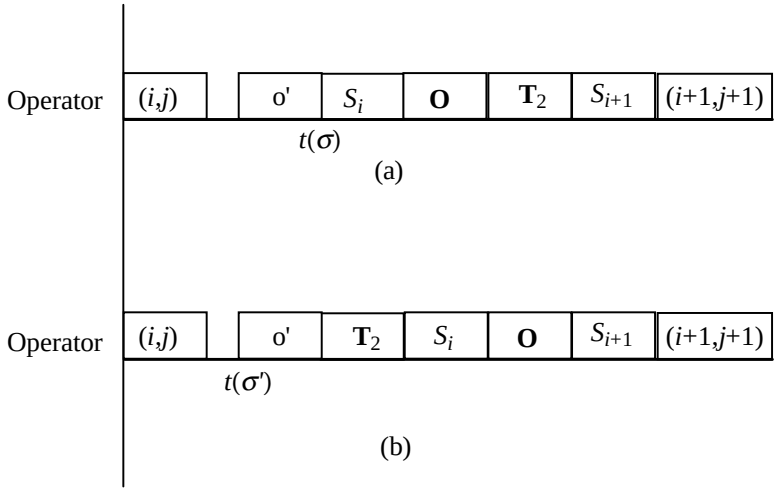

Figure 3.7: (a) A machine with positive inventory is set up; and (b) modification in case 2.

Define operation o' as follows: if case 1 applies, o' = operation (i, j); if case 2 applies, o' = the operation which is processed in σ immediately before the k-th setup of machine i. Observe that the set of operations processed after $t(\sigma')$ in σ' comprises all the operations processed after $t(\sigma)$ in σ together with some other operations that include at least operation o'. Hence, we need to apply the above rescheduling at most $\sum\sum n_{ij}$ times to obtain a schedule that satisfies the result stated in the Theorem. ■

Remark 3.1 *For any* $\delta \in \{L_{\max}, \sum C_j, \sum w_j C_j, \sum U_j, \sum w_j U_j\}$ *the result (ii) in Theorem 3.4 is false if the machine environment is a flow shop,* $m \geq 3$ *and the setup times are sequence-dependent. This is shown in Example 3.2 below.*

Example 3.2 *For any* $\delta' \in \{L_{\max}, \sum C_j, \sum w_j C_j, \sum U_j, \sum w_j U_j\}$ *consider an instance of* $1F3|s_{ii'}|\delta'$ *with* $s_{12} = s_{21} = s_{23} = s_{32} = 25$, $s_{13} = 1$, $s_{31} = 50$ *and the*

following processing times, due dates and weights:

j	1	2	3
t_{1j}	1	1	25
t_{2j}	1	80	80
t_{3j}	1	1	25
d_j	54	211	366
w_j	1	1	1

Observe that the setup times satisfy the triangle inequality condition. It follows from Theorem 3.3 that we can restrict the search for an optimal schedule to the schedules in which job-orders are same on all machines. The processing times, due dates and weights are all agreeable in the sense that $j < j' \Rightarrow t_{ij} \leq t_{ij'}\ \forall i$, $d_j \leq d_{j'}$ and $w_j \geq w_{j'}$. We consider the schedules in which jobs are processed in the ascending order of their indices.

Define schedule $\sigma^* = (1,1),(1,2),(2,1),(3,1),(2,2),(1,3),(3,2),(2,3),(3,3)$. In schedule σ^*, machine 1 is set up to process $(1,3)$ when the machine has a positive inventory containing job 2. Hence, σ^* violates condition (ii) of Theorem 3.4. Schedules in which a machine with positive inventory is not set up are as follows:

$\sigma_1 = (1,1),(2,1),(3,1),(1,2),(2,2),(3,2),(1,3),(2,3),(3,3)$

$\sigma_2 = (1,1),(2,1),(3,1),(1,2),(1,3),(2,2),(3,2),(2,3),(3,3)$

$\sigma_3 = (1,1),(2,1),(3,1),(1,2),(1,3),(2,2),(2,3),(3,2),(3,3)$

$\sigma_4 = (1,1),(1,2),(2,1),(3,1),(2,2),(3,2),(1,3),(2,3),(3,3)$

$\sigma_5 = (1,1),(1,2),(2,1),(2,2),(3,1),(3,2),(1,3),(2,3),(3,3)$

$\sigma_6 = (1,1),(1,2),(1,3),(2,1),(3,1),(2,2),(3,2),(2,3),(3,3)$

$\sigma_7 = (1,1),(1,2),(1,3),(2,1),(3,1),(2,2),(2,3),(3,2),(3,3)$

$\sigma_8 = (1,1),(1,2),(1,3),(2,1),(2,2),(3,1),(3,2),(3,2),(3,3)$

$\sigma_9 = (1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)$

In Table 3.1, we show the completion times of various jobs under various schedules. It can be observed from the completion times that σ^* is better than each schedule $\sigma_1, ..., \sigma_9$ with respect to any of $L_{\max}$, $\sum(w_j)C_j$, and $\sum(w_j)U_j$. Hence, condition (ii) of Theorem 3.4 is false if the machine environment is a flow shop, $m \geq 3$ and the setup times are sequence-dependent.

Schedule	C_1	C_2	C_3	$\sum(w_j)C_j$	$L_{\max}$	$\sum(w_j)U_j$
σ^*	54	211	366	631	0	0
σ_1	53	235	465	753	99	2
σ_2	53	260	415	728	49	2
σ_3	53	340	365	758	129	1
σ_4	54	185	415	654	49	1
σ_5	134	135	365	634	80	1
σ_6	79	210	365	654	25	1
σ_7	79	290	315	684	79	2
σ_8	159	160	315	634	105	1
σ_9	239	240	265	744	185	2

Table 3.1: Performance of some schedules

Remark 3.2 *The result (ii) in Theorem 3.4 is false if not all jobs are available at time zero. For example, if a very important job arrives at the shop floor so that the operator must complete the job immediately, it may make sense to incur as many setups as necessary to complete the job regardless of the inventory situation.*

Remark 3.3 *For $1Fm|s_i|\delta$, the condition that a machine with positive inventory is not set up implies that every switch from machine m is to the least indexed machine with no inventory and the switch takes place only after processing all jobs on machine m that are processed on all the other machines.*

Proof: If machine m is left before processing at least one job j which is processed on all the other machines, then the operator must set up a machine which has a positive inventory including at least job j. After leaving machine m, a machine i with no inventory is to be set up. The shop flow condition implies that machine i must have the least index among the ones with no inventory. ■

As we have discussed in Section 2.2, Coffman et al. [30] show that for a problem equivalent to $1O2|s_1 = s_2,\ t_{ij} = t_i|\sum C_j$ the search for an optimal schedule can be narrowed down to the schedules in which the operator switches from machine i to i' only if the number of jobs processed on machine i is strictly more than the number of jobs processed on machine i'. Julien [63] extends the result to the case of m machines with $m \geq 2$. In [63] he observes that for the open shop case, if every job has at least one operation on each machine, then it is sufficient to consider only those schedules in which every batch of operations, except the last one, creates inventory. In the following, we shall show that the result can be further extended to the open sequence-dependent cases and flow shop sequence-independent cases. Our proof is different from the proof given by Julien [63].

A similar result holds even if we assume that some job may not have any operation on some machine at all. In such a case, if the operator switches from machine i_1 to i_2 at time t, and if the first job completed after time t is job j^*, then it is better to complete operation (i_1, j^*) before switching, if operation (i_1, j^*) exists and if the shop flow condition is not violated by processing operation (i_1, j^*) before switching. We omit the proof of this statement here.

Theorem 3.5 *For both problems $1Om|s_{ii'}|\delta$ and $1Fm|s_i|\delta$ there exists an optimal schedule in which (i) job-orders are the same on all machines; (ii) a machine with positive inventory is not set up; and (iii) every switch is made after building a positive inventory unless both the following conditions hold: (a) the machine environment is a flow shop; and (b) the switch is from machine m.*

Proof: It follows from Theorem 3.4 that there exists an optimal schedule which satisfies results (i) and (ii) stated in the Theorem. Consider any schedule σ which satisfies results (i) and (ii) but not result (iii). By relabelling, if necessary, we may assume that the jobs are processed in the order 1, 2, ..., n on all machines. Since result (iii) does not hold, there exist i_1, i_2 and j^* such that a switch from machine i_1 to i_2 is immediately followed by operation (i_2, j^*) and such a switch takes place when operation (i_1, j^*) is unprocessed; we have $1 \leq i_1 \leq m$ in the case of an open shop and $1 \leq i_1 < m$ in the case of a flow shop.

Let $t(\sigma)$ be the minimum time when the operator leaves machine i_1, starts a setup of machine i_2 and then immediately processes job j^* on machine i_2 such that operation (i_1, j^*) is not processed before $t(\sigma)$. Result (i) implies that each operation (i_2, j), $j < j^*$ is processed sometime before $t(\sigma)$ and result (ii) implies that for each job $j < j^*$, completion time of job j, $C_j \leq t(\sigma)$. As operation (i_1, j^*) is not processed before $t(\sigma)$, no job can be completed after $t(\sigma)$ and before the next setup of machine i_1 and processing of operation (i_1, j^*). Let $t'(\sigma)$ be the time when operation (i_1, j^*) starts. A setup, say k^*-th setup of machine i_1, is completed at time $t'(\sigma)$ and for each job $j \geq j^*$, completion time of job j, $C_j \geq t'(\sigma) + t_{i_1,j^*}$. Let $\mathbf{T}$ be the set of operations and setups between $t(\sigma)$ and $t'(\sigma)$ and $\mathbf{O}$ be the batch of operations which contains (i_1, j^*). Let $\mathbf{T}'$ be the set of operations and setups that are processed after the completion of operation (i_2, j^*) and before the start of the k^*-th setup on machine i_1. Let $\mathbf{O}'$ be the set of operations in $\mathbf{O}$ except operation (i_1, j^*).

Case 1a: Suppose that (i_1, j^*) is the last operation of job j^* and $|\mathbf{O}| > 1$. Remove operation (i_1, j^*), delay all operations and setups in $\mathbf{T}$ by t_{i_1,j^*} time units and schedule operation (i_1, j^*) at time $t(\sigma)$.

Case 1b: Suppose that (i_1, j^*) is the last operation of job j^* and $|\mathbf{O}| = 1$. Remove operation (i_1, j^*), delay all operations and setups in $\mathbf{T}$ by t_{i_1,j^*} time units and schedule operation (i_1, j^*) at time $t(\sigma)$. Remove the k^*-th setup of machine i_1. Perform left-shift operations to obtain a semi-active schedule.

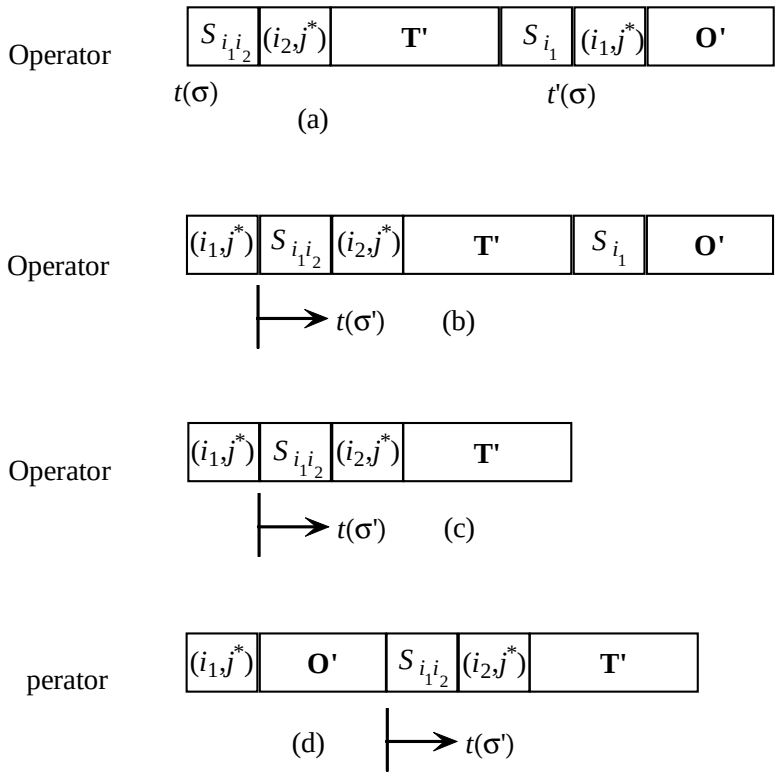

Figure 3.8: (a) A switch is made without building a positive inventory; (b) modification in case 1a; (c) modification in case 1b; and (d) modification in case 2.

Case 2: Suppose that (i_1, j^*) is not the last operation of job j^*. Result (i) implies that no operation $(i_1, j) \in \mathbf{O}$ can be the last operation of job j. Remove all operation $(i_1, j) \in \mathbf{O}$ and delay all operations and setups in $\mathbf{T}$ by $\sum_{(i_1,j)\in\mathbf{O}} t_{i_1,j}$ time units. Insert all operations in $\mathbf{O}$ between time $t(\sigma)$ and $t(\sigma) + \sum_{(i_1,j)\in\mathbf{O}} t_{i_1,j}$ and process the operations in the order in which they are processed in σ. Remove the k^*-th setup of machine i_1. Perform left-shift operations to obtain a semi-active schedule.

Call the modified schedule σ'. In cases 1a and 1b $t(\sigma') \geq t(\sigma) + t_{i_1,j^*}$ and in case 2 $t(\sigma') \geq t(\sigma) + \sum_{(i_1,j)\in\mathbf{O}} t_{i_1,j}$. If σ' does not satisfy the result stated in the Theorem, set $\sigma \leftarrow \sigma'$ and repeat the procedure.

Figure 3.8 illustrates an iteration. Consider any iteration. In cases 1a and 1b we rearrange the operations and setups processed between $t(\sigma)$ and $t'(\sigma) + t_{i_1,j^*}$. But, no job is completed between $t(\sigma)$ and $t'(\sigma) + t_{i_1,j^*}$ in schedule σ. In case 2 we rearrange operations and setups processed between $t(\sigma)$ and $t'(\sigma) + \sum_{(i_1,j)\in\mathbf{O}} t_{i_1,j}$. Again, no job is completed between $t(\sigma)$ and $t'(\sigma) + \sum_{(i_1,j)\in\mathbf{O}} t_{i_1,j}$ in schedule σ. Hence, such rearrangement does not increase any completion time. In cases 1b and 2 we remove the k^*-th setup of machine i_1. From the triangle inequality condition satisfied by the setup times, completion times do not increase by removing a setup. Hence, no completion time increases.

In cases 1a and 1b we move operation (i_1, j^*) ahead of $\mathbf{T}$ and in case 2 we move the entire batch $\mathbf{O}$ of operations on machine i_1 ahead of $\mathbf{T}$. In no case may $\mathbf{T}$ contain

any operation on machine i_1. Hence, the job-order on every machine is the same in both σ and σ'. This shows that σ' satisfies result (i).

We shall now show that σ' satisfies result (ii). Let $I_{i,k}(\sigma)$ and $I_{i,k}(\sigma')$ be the number of jobs in the inventory of machine i at the time of the k-th setup of machine i in schedule σ and σ' respectively. Since σ satisfies result (ii), $I_{i,k}(\sigma) = 0$ for all i and k. We shall now show that $I_{i,k}(\sigma') = 0$ for all i and k.

Case A: Consider any k and $i \neq i_1$. *If the k-th setup of machine i is not in* $\mathbf{T}$, the set of operations completed before the k-th setup of machine i in schedule σ' and the set of operations completed before the k-th setup of machine i in schedule σ are same. Hence, $I_{i,k}(\sigma') = I_{i,k}(\sigma) = 0$. *If the k-th setup of machine i is in* $\mathbf{T}$, the set of operations completed before the k-th setup of machine i in schedule σ' comprises all the operations completed before the k-th setup of machine i in schedule σ and some other operations which are moved ahead of $\mathbf{T}$. But, each operation moved ahead of $\mathbf{T}$ is processed on machine i_1. Hence $I_{i,k}(\sigma') \leq I_{i,k}(\sigma) = 0$.

Case B: Consider machine i_1 and $k < k^*$. The k-th setup of machine i_1 takes place before time $t(\sigma)$ and the partial schedule up to time $t(\sigma)$ is the same in both σ and σ'. Hence, $I_{i_1,k}(\sigma') = I_{i_1,k}(\sigma) = 0$.

Case C: Consider machine i_1 and $k = k^*$. *If case 1a applies,* the set of operations completed before the k^*-th setup of machine i_1 in schedule σ' comprises all the operations completed before the k^*-th setup of machine i in schedule σ and operation

(i_1, j^*). Hence, $I_{i_1,k^*}(\sigma) \leq I_{i_1,k^*}(\sigma') \leq I_{i_1,k^*}(\sigma) + 1$. From the assumption of case 1a, operation (i_1, j^*) is the last operation of job j^* in schedule σ. Since schedule σ' is obtained from schedule σ by moving the operation (i_1, j^*) ahead of $\mathbf{T}$, the last operation of job j^* in schedule σ' is in $\mathbf{T}$. (Note that at least (i_2, j^*) is in $\mathbf{T}$. Hence, the last operation of job j^* is precisely in $\mathbf{T}$). This means that operation (i_1, j^*) is completed before the k^*-th setup of machine i_1. Then, $I_{i_1,k^*}(\sigma') \neq I_{i_1,k^*}(\sigma) + 1$. Hence, $I_{i_1,k^*}(\sigma') = I_{i_1,k^*}(\sigma) = 0$. *If case 1b or 2 applies*, the k^*-th setup of machine i_1 exists in σ' iff the $(k^* + 1)$-st setup of machine i_1 exists in σ. Furthermore, the set of operations completed before the k^*-th setup of machine i_1 in schedule σ' and the set of operations completed before the $(k^* + 1)$-st setup of machine i_1 in schedule σ are same. Hence, $I_{i_1,k^*}(\sigma') = I_{i_1,k^*+1}(\sigma) = 0$.

Case D: Consider machine i_1 and $k > k^*$. *If case 1a applies*, the k-th setup of machine i_1 exists in σ iff the k-th setup of machine i_1 exists in σ'. Furthermore, the set of operations completed before the k-th setup of machine i_1 in schedule σ' and the set of operations completed before the k-th setup of machine i_1 in schedule σ are same. Hence, $I_{i_1,k}(\sigma') = I_{i_1,k}(\sigma) = 0$. *If case 1b or 2 applies*, the k-th setup of machine i_1 exists in σ' iff the $(k + 1)$-st setup of machine i_1 exists in σ. Furthermore, the set of operations completed before the k-th setup of machine i_1 in schedule σ' and the set of operations completed before the $(k + 1)$-st setup of machine i_1 in schedule σ are same. Hence, $I_{i_1,k}(\sigma') = I_{i_1,k+1}(\sigma) = 0$.

Hence schedule σ' satisfies condition (ii).

Observe that the set of operations processed before $t(\sigma')$ in σ' comprises all the operations processed before $t(\sigma)$ in σ and some other operations that include at least operation (i_1, j^*). Hence, the number of operations processed before $t(\sigma')$ in σ' is at least one more than the number of operations processed before $t(\sigma)$ in σ. This means that we need to apply the above rescheduling at most mn times to obtain a schedule that satisfies the condition stated in the Theorem. ■

3.3 Batching Schedules

In this subsection, we shall define the concept of a batching schedule. Then, in the next subsection we shall show that in the case of two machines, an optimal batching schedule is an optimal schedule for any regular objective.

Definition 3.1 *A batch of jobs is a maximal set* $\mathbf{B}$ *of jobs satisfying the following conditions (i) on each machine* i*, all operations* (i, j)*,* $j \in \mathbf{B}$ *are processed in at most one setup; (ii) For any job* $j \notin \mathbf{B}$*, and any machine* i*, the operation* (i, j) *is scheduled either before the first scheduled operation of* $\mathbf{B}$ *or after the last scheduled operation of* $\mathbf{B}$.

Essentially, a batch is a group of jobs that behaves like a single "combined" job and is processed from start to finish in the shop without being pre-empted by other jobs not in the batch.

In the three-job schedule $\sigma_1 = (2,3),(2,2),(2,1),(1,1),(1,2),(1,3)$, jobs 1, 2 and 3 constitute a single batch. Each set $\{1\}$ and $\{1,2\}$ satisfies conditions (i) and (ii) but none of $\{1\}$ and $\{1,2\}$ is a maximal set of jobs satisfying condition (i) and (ii). Hence, none of $\{1\}$ and $\{1,2\}$ is a batch.

A batch $\mathbf{B}$ may pre-empt a job $j \notin \mathbf{B}$. For example, in the two-job schedule $\sigma_2 = (1,1),(2,2),(1,2),(2,1)$ set $\{2\}$ is a batch which pre-empts job 1. Job 1 is processed on machine 1 before batch $\{2\}$ and on machine 2 after batch $\{2\}$.

In general, a schedule may have no batch at all. For example, consider the two-job schedule $\sigma_3 = (1,1),(2,2),(2,1),(1,2)$. Job 1 does not constitute a batch because operation (2,2) is processed between two operations of job 1. Job 2 does not constitute a batch because operation (2,1) is processed between two operations of job 2. Jobs 1 and 2 do not constitute a batch because jobs 1 and 2 are not processed in a single setup on machine 1.

Henceforth, we shall use the term batch to mean batch of jobs.

Definition 3.2 *We say that $\psi_i(p)$ is the job-order on machine i, if $\psi_i(p)$ is the p-th processed job on machine i, for $p = 1,2,...,n$.*

Definition 3.3 *A schedule is a batching schedule if the job-order is the same on all machines and every job belongs to a batch.*

Consider a batching schedule with job-order ψ. Every batch is of the type $\{\psi(p), \psi(p+1), ..., \psi(p')\}$ for some $1 \leq p \leq p' \leq n$. We shall denote such a batch by $[p,p']$. If

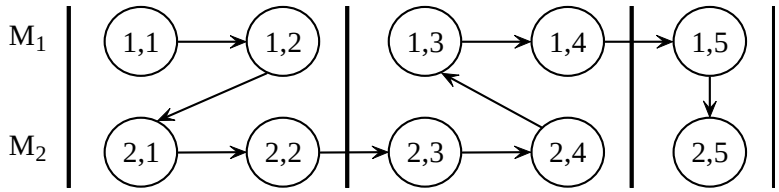

Figure 3.9: A batching schedule

the job-order is known and fixed we may assume, by relabelling if necessary, that the job-order is 1, 2, ..., n. In this ordering, batch $[i,j]$ is $\{i, i+1, \ldots, j\}$. For example, the batch in schedule σ_1 above is $[1,3]$. Figure 3.9 shows a batching schedule with batches $[1,2]$, $[3,4]$ and $[5,5]$.

Consider a batching schedule with job-order ψ and batches $\mathbf{B}_1$, $\mathbf{B}_2$, ..., $\mathbf{B}_k$. Let $|\mathbf{B}|$ denote the cardinality of a set $\mathbf{B}$, and define $p_0 = 0$ and $p_u = \sum_{v=1}^{u} |\mathbf{B}_v|$ for each u, $1 \leq u \leq k$. Then, the u-th batch is $\mathbf{B}_u = \{\psi(p_{u-1}+1), \psi(p_{u-1}+2), \ldots, \psi(p_u)\}$. Hence, the batches can be constructed from the job-order ψ and integers p_1, p_2, ..., p_k.

Definition 3.4 *A batching schedule with k batches and job-order ψ has a batching policy $\mu = (p_1, p_2, \ldots, p_k)$ if, for $u = 1, 2, \ldots, k$ the u-th batch is $\mathbf{B}_u = \{\psi(p_{u-1}+1), \psi(p_{u-1}+2), \ldots, \psi(p_u)\}$, where $p_0 = 0$.*

In Figure 3.9 the batches are separated by vertical lines. If every job belongs to a batch, all operations of a batch are contiguous with the exception of some setups. For a batch $\mathbf{B}$, an operation (i,j), $j \notin \mathbf{B}$ is not processed between the first and last

operation of batch **B**. Hence, such vertical lines can be used to separate batches.

An interpretation of the batching policy is that it indicates where the vertical lines are drawn. The leftmost and rightmost vertical lines are always drawn. The other vertical lines depend on the batching policy. Consider a batching policy $\mu = (p_1, p_2, ..., p_k)$. For each $u = 1, 2, ..., (k-1)$ nodes corresponding to jobs $\psi(p_u)$ and $\psi(p_u+1)$ are separated by a vertical line. The schedule shown in Figure 3.9 has a job-order 1, 2, ..., 5 and a batching policy $\mu = (2, 4, 5)$. One vertical line separates nodes corresponding to jobs 2 and 3 and another vertical line separates nodes corresponding to jobs 4 and 5.

3.4 Optimality of Batching Schedules

For the case of two machines and any regular objective, it is sufficient to limit the search for an optimal schedule to the batching schedules.

Theorem 3.6 *For both problems* $1O2|s_i|\delta$ *and* $1F2|s_i|\delta$*, there exists an optimal schedule which is a batching schedule. For the flow shop case, every batch starts on machine 1. For the open shop case, two consecutive batches start on different machines.*

Proof: Consider a schedule σ that satisfies conditions stated in Theorem 3.5. Job-orders are same on all machines. Suppose that σ involves a total of K setups. Each odd setup is followed by an operation on machine i_1 and each even setup is followed

by an operation on machine $i_2 \neq i_1$ where $i_1 \in \{1, 2\}$ in the open shop case and $i_1 = 1$ in the flow shop case.

Every job has exactly one operation on each machine, so requires two setups. In fact, every job is processed in two consecutive setups. For, if this is not true, there exists a job j which is processed on machine i in the u-th setup and on machine $i' \neq i$ in the v-th setup, with $v > u+2$. But then, each of the u'-th setups with $u' \in \{u+2, u+4, ..., v-1\}$ is on machine i, and the u'-th setup takes place when machine i has a positive inventory containing at least job j. But, by the choice of σ, a machine with positive inventory is not set up. This contradiction proves the statement that every job is processed in two consecutive setups.

Let $\mathbf{B}_k$ be the set of jobs processed in setups k and $(k+1)$.

In the flow shop case, $\mathbf{B}_k = \emptyset$ for even k, so σ is a batching schedule with batches $\mathbf{B}_1, \mathbf{B}_3, ..., \mathbf{B}_{K-1}$. This completes the proof for the flow shop case.

To complete the proof for the open shop case we show that $\mathbf{B}_k \neq \emptyset$ for any k. Suppose that for some k, the k-th setup is on machine i and $\mathbf{B}_k = \emptyset$. Notice that at least one job is processed in the k-th setup, because otherwise, setups k and $(k+1)$ are contiguous and the k-th setup can be eliminated. Since all the jobs are processed in two consecutive setups and $\mathbf{B}_k = \emptyset$, all the jobs processed in the k-th setup are processed in setup $(k-1)$. This means that the operator does not build any inventory on machine i before setup $(k+1)$ which is on machine $i' \neq i$. But, by the choice of σ,

every switch is made after building a positive inventory. Hence, $\mathbf{B}_k \neq \emptyset$ for all k, so σ is a batching schedule with batches $\mathbf{B}_k$, $1 \leq k < K$. This completes the proof for the open shop case. ■

As we have discussed in Section 2.2, Coffman et al. [30] show that for a problem equivalent to $1O2|s_1 = s_2,\ t_{ij} = t_i|\sum C_j$ the search for an optimal schedule can be narrowed down to the schedules in which the operator switches from machine i to i' only if the number of jobs processed on machine i is strictly more than the number of jobs processed on machine i'. Observe that Theorem 3.6 has exactly the same implication for the problem $1O2|s_1 = s_2,\ t_{ij} = t_i|\sum C_j$. Theorem 3.6 explains the observations of Baker [12], Julien and Magazine [64], Vickson et al. [117], Sung and Park [111], and Rana and Singh [98] on various two-machine cases with the objective of $\sum C_j$.

We shall now show that for $\delta' \in \{L_{\max}, \sum C_j, \sum w_j C_j, \sum U_j, \sum w_j U_j\}$ and $m > 2$ it is *not* sufficient to limit the search for an optimal schedule to the batching schedules.

Example 3.3 *Consider an instance of* $1O3|s_i|\delta'$ *or* $1F3|s_i|\delta'$ *with* $s_1 = 10, s_2 = 5, s_3 = 0$ *and the following processing times, due dates and weights:*

j	1	2	3
t_{1j}	1	1	1
t_{2j}	1	1	15
t_{3j}	5	10	15
d_j	25	35	70
w_j	1	1	1

Consider any batching schedule σ with job-order ψ. Before time $C_{\psi(1)}$, each machine has to be set up at least once and all operations of a job have to be completed. Hence, $C_{\psi(1)} \geq 22$. Before time $C_{\psi(2)}$, each machine has to be set up at least once and all operations of any two jobs have to be completed. Hence, $C_{\psi(2)} \geq 34$. Before time $C_{\psi(3)}$, each machine has to be set up at least once and all operations have to be completed. Hence, $C_{\psi(3)} \geq 65$.

Let $\sigma^* = (1,1), (1,2), (1,3), (2,1), (2,2), (3,1), (3,2), (2,3), (3,3)$. The schedule maintains the shop flow condition and attains $C_1 = 25$, $C_2 = 35$ and $C_3 = 70$. Therefore, $L_{\max} = 0$, $\sum w_j C_j = 130$ and $\sum w_j U_j = 0$. We shall now show that for any δ', every batching schedule is worse than σ^*.

Case 1: Suppose that σ has a single batch. Each machine is set up once and $C_{\psi(3)} = 65$. At most one operation is completed after $C_{\psi(2)}$, and the largest operation has a processing time of 15 time units. Hence, $C_{\psi(2)} \geq 65 - 15 = 50$. Therefore, schedule σ is worse than σ^* for any δ'.

Case 2: Suppose that σ has two batches. Consider first the flow shop case. Each machine is set up twice. Hence, $C_{\psi(3)} \geq 80$ and, therefore, schedule σ is worse than σ^* for any δ'. Consider next the open shop case. One machine is set up once and each of the others twice. If machine 1 is set up twice, $C_{\psi(3)} \geq 75$ and, therefore schedule σ is worse than σ^* for any δ'. Suppose, instead, that machine 1 is set up once and each of the others twice. Then, $C_{\psi(3)} \geq 70$. If the first batch contains two jobs j_1 and j_2,

all four operations of jobs j_1 and j_2 on machines 2 and 3 precede $C_{\psi(1)}$. Hence, $C_{\psi(1)} \geq 33$. If the first batch does not contain two jobs, each of the machines 2 and 3 is set up twice before $C_{\psi(2)}$. Hence, $C_{\psi(2)} \geq 39$. Since $C_{\psi(1)} \geq 33$, $C_{\psi(2)} \geq 39$ and $C_{\psi(3)} \geq 70$, schedule σ is worse than σ^* for any δ'.

Case 3: Suppose that σ has three batches. Before $C_{\psi(2)}$ one machine is set up at least once and each of the other two twice. Hence, $C_{\psi(2)} \geq 39$. Before $C_{\psi(3)}$ one machine is set up at least twice and each of the other two three times. Hence, $C_{\psi(3)} \geq 85$. Therefore, schedule σ is worse than σ^* for any δ'.

3.5 A Relationship between $1F2|s_i|\delta$ and $1|s_1, F = 1|\delta$

The single-family problem $1|s_1, F = 1|\delta$ is a special case of the problem $1F2|s_i|\delta$. To see this, consider an n-job instance $\mathcal{I}_1$ of the problem $1|s_1, F = 1|\delta$ with setup time s_1. For each job j let t_j be the processing time, w_j be the weight and d_j be the due date. Define an instance $\mathcal{I}_2$ of the problem $1F2|s_i|\delta$ with setup times $s'_1 = s_1$, $s'_2 = 0$, processing times $t'_{1j} = t_j$, $t'_{2j} = 0$, weights $w'_j = w_j$ and due dates $d'_j = d_j$, where $j = 1, ..., n$.

Consider a schedule σ_1 of $\mathcal{I}_1$ with batches $\mathbf{B}_1, ..., \mathbf{B}_k$, where $u_1 < u_2 \Rightarrow \mathbf{B}_{u_1}$ is processed before $\mathbf{B}_{u_2}$. The completion time of any job $j \in \mathbf{B}_u$ is $C_j = s_1 u + \sum(t_{j'} : j' \in \mathbf{B}_1 \cup ... \cup \mathbf{B}_u)$. Now consider a schedule σ_2 of $\mathcal{I}_2$ with batches $\mathbf{B}_1, ..., \mathbf{B}_k$, where $u_1 < u_2 \Rightarrow \mathbf{B}_{u_1}$ is processed before $\mathbf{B}_{u_2}$. The completion time of any job $j \in \mathbf{B}_u$ is C'_j

$= s'_1 u + \sum(t'_{1j'} : j' \in \mathbf{B}_1 \cup ... \cup \mathbf{B}_u) = C_j$. It follows that for any regular objective δ, the objective function value $\delta(\sigma_1) = \delta(\sigma_2)$. This shows that the problem $1|s_1, F = 1|\delta$ is a special case of the problem $1F2|s_i|\delta$.

This result means that any algorithm for the problem $1F2|s_i|\delta$ can be applied to solve the problem $1|s_1,\ F = 1|\delta$, and if the problem $1|s_1,\ F = 1|\delta$ is (strongly) $\mathcal{NP}$-hard for some δ then the problem $1F2|s_i|\delta$ is also (strongly) $\mathcal{NP}$-hard. Hochbaum and Landy [58] show that the problem $1|s_1,\ F = 1|\sum w_j U_j$ is $\mathcal{NP}$-hard, hence so is the problem $1F2|s_i|\sum w_j U_j$. Albers and Brucker [4] show that the problem $1|s_1,\ F = 1|\sum w_j C_j$ is strongly $\mathcal{NP}$-hard, hence so is the problem $1F2|s_i|\sum w_j C_j$.

3.6 Summary

Recall from our statement of the one-operator scheduling problem in Section 1.2 that every job has *at least one* operation on each machine. It follows from Theorem 3.1 that we can replace the phrase "at least one" by "exactly one" in the preceding statement and yet get an equivalent problem.

Developments on m-machine cases culminate in Theorem 3.5. For both problems $1Om|s_{ii'}|\delta$ and $1Fm|s_i|\delta$ there exists an optimal schedule in which (i) job-orders are the same on all machines; (ii) a machine with positive inventory is not set up; and (iii) every switch is made after building a positive inventory unless both the following conditions hold: (a) the machine environment is a flow shop; and (b) the switch is

from machine m.

Results (i) and (ii) are similar to what Santos [106] observes for a problem equivalent to $1Fm|s_i, t_{ij} = t_i|\sum C_j$. Coffman et al. [30] and Julien [63] observe results similar to the ones stated in Theorem 3.5 for problems equivalent to $1O2|s_1 = s_2$, $t_{ij} = t_i|\sum C_j$ and $1Om|s_i, n_{ij} \geq 0|\sum C_j$, respectively. Julien [63] states that the properties extend to any regular objective. Julien's proofs in [63] are different from ours.

The dominance properties developed for the m-machine cases are used in Chapter 4 to develop a dynamic programming scheme that solves a large number of cases we consider.

The dominance properties developed for the m-machine cases have a nice interpretation for the case of two machines. We can narrow down the search for an optimal schedule to batching schedules. A batching schedule is defined by specifying the first machine on which the operator starts working, a job-order and a batching policy. Developments on Chapters 5-8 are based on this result on two machines. For the case of three or more machines, it is not sufficient to consider the batching schedules only.

In this chapter we show that the problem $1|s_1, F = 1|\delta$ is a special case of the problem $1F2|s_i|\delta$. The observation is important because an immediate consequence of this is that the problem $1F2|s_i|\sum w_jU_j$ is $\mathcal{NP}$-hard (follows from a result in [58]) and the problem $1F2|s_i|\sum w_jC_j$ is strongly $\mathcal{NP}$-hard (follows from a result in [4]).

Chapter 4

The Fixed-Sequence Case

It follows from Theorem 3.3 that we can restrict the search for an optimal schedule to the cases in which job-orders are same on all machines. In this chapter we shall concentrate on what we call the fixed-sequence cases. In a fixed-sequence case, job-orders on all the machines are known and fixed. The importance of such cases is that for some objectives and job characteristics it is possible that one job-order dominates all the other job-orders.

As we have discussed in Section 2.1.2, Psaraftis [97], Ahn and Hyun [3] and Ghosh [47] use similar dynamic programming recursions for various cases of the problem $1|s_{ii'}|\sum(w_j)C_j$. Ghosh and Gupta [48] extend the approach to the problem $1|s_{ii'}|L_{\max}$. In this chapter we shall use a similar approach and apply the dominance properties discussed in chapter 3 in order to develop a common dynamic programming scheme that applies to many fixed-sequence cases.

First, we shall discuss some of the cases in which we get dominant job-orders.

4.1 Dominant Job-Orders

For both problems $1Om|s_{ii'}|C_{\max}$ and $1Fm|s_{ii'}|C_{\max}$ there exists an optimal schedule in which each machine is set up exactly once and jobs are processed in any order. To see this, consider any schedule in which a machine i is set up more than once. Move all the jobs processed in the second setup of machine i to immediately after the first setup of machine i and remove the second setup of machine i. The makespan does not increase. Repeated application of this process yields a schedule in which each machine is set up exactly once.

Rearranging jobs on a machine does not change the makespan. Thus, every job-order is equally good for the makespan objective. The issue, however, is to minimize the total setup times required. Since all the jobs are processed on each machine in a single setup, we can limit our discussion to the case with $n = 1$, if the objective is makespan. If $n > 1$, we can replace the jobs with a single job having processing time $t_i = \sum_{j=1}^{n} t_{ij}$ on machine i.

Theorem 4.1 *For both problems $1Om|s_{ii'}|L_{\max}$ and $1Fm|s_{ii'}|L_{\max}$ there exists an optimal schedule in which the jobs are processed in the Earliest Due Date (EDD) order on each machine.*

Proof: Consider any schedule σ with the same job-order on all machines. By relabelling, if necessary, we may assume that $j < j' \Rightarrow d_j \leq d_{j'}$. If the jobs are not

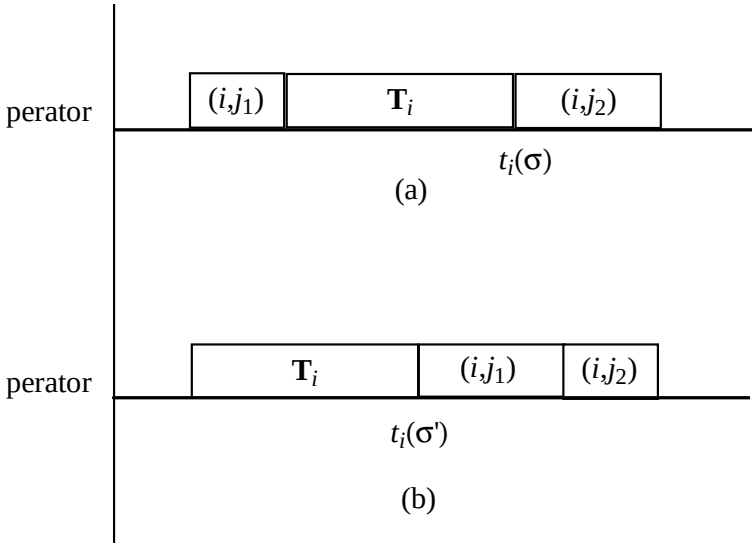

Figure 4.1: (a) Two operations on a same machine are not in the order 1,2,...,n; and (b) a modified schedule.

processed in the EDD order, then there exist two jobs j_1 and j_2 such that $j_1 > j_2$ but j_1 is completed before j_2. We shall repeatedly apply the following rescheduling procedure (see Figure 4.1) to obtain a schedule which is not worse than the original schedule, but satisfies the condition stated in the Theorem. Throughout, we maintain the same job-order on all machines.

For any job j, let C_j be its completion time in schedule σ. Let $t(\sigma)$ be the maximum time when a set $\mathbf{T}$ of operations and setups is completed, such that: $\mathbf{T}$ starts immediately after (i', j_1) and is completed immediately before (i', j_2), where $j_1 > j_2$ and j_1 is processed before j_2 on each machine. Since the job-order is the same on each machine, we have for each machine i a set $\mathbf{T}_i$ of operations and setups such that

$\mathbf{T}_i$ is processed after operation (i, j_1) and before operation (i, j_2). Let the completion time of $\mathbf{T}_i$ be $t_i(\sigma)$. We thus have $t_i(\sigma) \leq t(\sigma)\ \forall i$. For each machine i, remove operation (i, j_1), schedule every operation and setup in $\mathbf{T}_i \cup \{(i, j_2)\}$ to start t_{i,j_1} time units earlier, and then schedule the operation (i, j_1) to start at time $t_i(\sigma) + t_{i,j_2} - t_{i,j_1}$.

Call the modified schedule σ'. For every job j, let C'_j be its completion time in schedule σ'. Notice that the last operation of job j_2 is on machine i' in schedule σ. Operation (i', j_1) is processed immediately after operation (i', j_2) in schedule σ'. The start times of operation (i', j_2) are $t(\sigma)$ and $t(\sigma) - t_{i',j_1}$ in schedules σ and σ', respectively. Hence, $t(\sigma') \leq t(\sigma) - t_{i',j_1}$. If σ' does not satisfy the condition stated in the Theorem, set $\sigma = \sigma'$ and repeat the procedure.

In every iteration, the completion time of operation (i, j_1) for each i may increase. However, $C'_{j_1} = C_{j_2}$. Letting L_j and L'_j be the lateness of job j in schedules σ and σ', respectively, we get $L'_{j_1} = C'_{j_1} - d_{j_1} = C_{j_2} - d_{j_1} \leq C_{j_2} - d_{j_2} = L_{j_2}$. Since $C'_j \leq C_j\ \forall j \neq j_1$, we have $L'_j \leq L_j\ \forall j \neq j_1$. Hence, $\max\{L'_j\} \leq \max\{L_j\}$.

Observe that the set of jobs completed after $t(\sigma')$ in σ' comprises all the jobs completed after $t(\sigma)$ in σ, together with some other jobs that include at least j_1. Hence, the number of jobs processed after $t(\sigma')$ in σ' is at least one more than the number of jobs processed after $t(\sigma)$ in σ. Hence, we need to apply the above rescheduling procedure at most n times to obtain a schedule that satisfies the condition stated in the Theorem. ■

Theorem 4.2 *For both problems* $1Om|s_{ii'}, \text{aptw}|\sum w_jC_j$ *and* $1Fm|s_{ii'}, \text{aptw}|\sum w_jC_j$ *there exists an optimal schedule in which the jobs are processed in the order 1, 2, ..., n.*

Proof: Consider any schedule σ with the same job-order on all machines. If the jobs are not processed in the order 1, 2, ..., n, then there exist two jobs j_1 and j_2 such that $j_1 > j_2$ but j_1 is processed before j_2. We shall repeatedly apply the following rescheduling procedure (see Figure 4.2) to obtain a schedule which is not worse than the original schedule, but satisfies the condition stated in the Theorem. Throughout, we maintain the same job-order on all machines.

For any job j, let C_j be the completion time of job j in schedule σ. Let $j_1(\sigma) = \max\{j_1 : \text{there exists at least one job } j_2 \text{ such that } j_2 < j_1 \text{ and } j_2 \text{ is processed after } j_1\}$. Let $j_2(\sigma) = \arg\max\{C_{j_2} : j_2 < j_1(\sigma)\}$. By the choice of $j_1(\sigma)$, $j_2(\sigma)$ is processed after $j_1(\sigma)$. Furthermore, $C_{j_1(\sigma)+1} \leq ... \leq C_n$ and each job $j \leq j_1(\sigma)$ is completed before $(j_1(\sigma) + 1)$.

Let σ' be the schedule obtained by interchanging operations $(i, j_1(\sigma))$ and $(i, j_2(\sigma))$ for each i. Let C'_j be the completion time of job j in schedule σ'. By the choice of $j_2(\sigma)$ there exists no job $j_2 < j_1(\sigma)$ such that j_2 is processed after $j_1(\sigma)$ in schedule σ'. Hence, $j_1(\sigma') < j_1(\sigma)$. If σ' does not satisfy the condition stated in the Theorem, set $\sigma = \sigma'$ and repeat the procedure.

Consider any iteration. Suppose that some operation (i_1, j_1) is the k-th processed

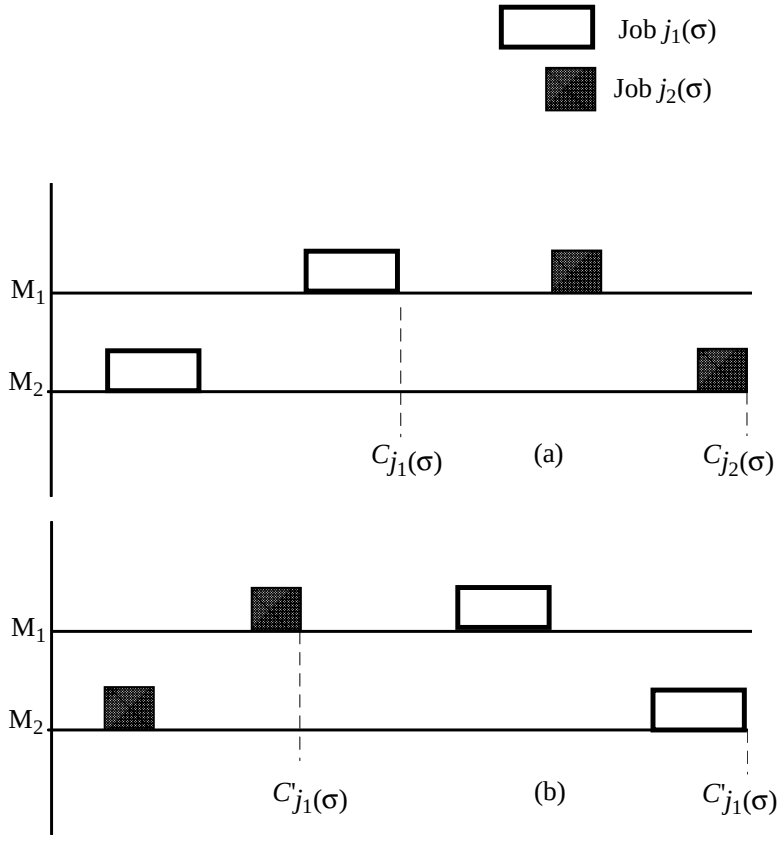

Figure 4.2: (a) Two operations on a same machine are not in the order 1,2,...,n; and (b) a modified schedule.

operation in schedule σ and (i_2, j_2) is the k-th processed operation in schedule σ'. We must have $i_2 = i_1$. If $j_1 = j_1(\sigma)$, then $j_2 = j_2(\sigma)$. If $j_1 = j_2(\sigma)$, then $j_2 = j_1(\sigma)$. If $j_1 \neq j_1(\sigma)$, and $j_1 \neq j_2(\sigma)$, then $j_2 = j_1$. In any case, if (i_1, j_1) is the last operation of job j_1 in schedule σ, then (i_2, j_2) is the last operation of j_2 in schedule σ'.

Let $\mathbf{M}_1 = \{i :$ either $(i, j_1(\sigma))$ is scheduled before (i_1, j_1) in schedule σ or $(i, j_1(\sigma)) \equiv (i_1, j_1)\}$. Let $\mathbf{M}_2 = \{i :$ either $(i, j_2(\sigma))$ is scheduled before (i_1, j_1) in schedule σ or $(i, j_2(\sigma)) \equiv (i_1, j_1)\}$. Since $j_1(\sigma)$ is processed before $j_2(\sigma)$ in schedule σ, we must have $\mathbf{M}_1 \supseteq \mathbf{M}_2$. Let $C'_{i_2 j_2}$ be the completion time of (i_2, j_2) in σ' and $C_{i_1 j_1}$ be the completion time of (i_1, j_1) in σ. Observe that

$$\begin{aligned} C'_{i_2 j_2} &= C_{i_1 j_1} + \sum_{i \in \mathbf{M}_1} t_{i j_2(\sigma)} - \sum_{i \in \mathbf{M}_1} t_{i j_1(\sigma)} + \sum_{i \in \mathbf{M}_2} t_{i j_1(\sigma)} - \sum_{i \in \mathbf{M}_2} t_{i j_2(\sigma)} \\ &= C_{i_1 j_1} + \sum_{i \in \mathbf{M}_1 \setminus \mathbf{M}_2} (t_{i j_2(\sigma)} - t_{i j_1(\sigma)}) \\ &\leq C_{i_1 j_1} \end{aligned}$$

In fact, the above inequality can be replaced by equality if (i_2, j_2) is the last operation of $j_1(\sigma)$ in σ'. In such a case (i_1, j_1) is the last operation of $j_2(\sigma)$ in σ and $\mathbf{M}_1 = \mathbf{M}_2 = \{1, 2, .., m\}$. Now, using the relationship between machines i_1 and i_2 and jobs j_1 and j_2, we have

$$C'_{j_1(\sigma)} = C_{j_2(\sigma)} \tag{4.1}$$

$$C'_{j_2(\sigma)} \leq C_{j_1(\sigma)} \tag{4.2}$$

$$C'_j \leq C_j \qquad \text{for } j \neq j_1(\sigma), j \neq j_2(\sigma). \tag{4.3}$$

Since $j_1(\sigma) > j_2(\sigma)$, we have $w_{j_1(\sigma)} \leq w_{j_2(\sigma)}$. But $C_{j_1(\sigma)} \leq C_{j_2(\sigma)}$. Hence,

$$w_{j_1(\sigma)}C_{j_2(\sigma)} + w_{j_2(\sigma)}C_{j_1(\sigma)} \leq w_{j_2(\sigma)}C_{j_2(\sigma)} + w_{j_1(\sigma)}C_{j_1(\sigma)}. \tag{4.4}$$

Therefore,

$$\begin{array}{rcll}
\sum w_j C'_j & = & \sum_{j\neq j_1(\sigma) \text{ and } j\neq j_2(\sigma)} w_j C'_j + w_{j_1(\sigma)}C'_{j_1(\sigma)} + w_{j_2(\sigma)}C'_{j_2(\sigma)} & \\
& = & \sum_{j\neq j_1(\sigma) \text{ and } j\neq j_2(\sigma)} w_j C'_j + w_{j_1(\sigma)}C_{j_2(\sigma)} + w_{j_2(\sigma)}C'_{j_2(\sigma)} & \text{from 4.1} \\
& \leq & \sum_{j\neq j_1(\sigma) \text{ and } j\neq j_2(\sigma)} w_j C'_j + w_{j_1(\sigma)}C_{j_2(\sigma)} + w_{j_2(\sigma)}C_{j_1(\sigma)} & \text{from 4.2} \\
& \leq & \sum_{j\neq j_1(\sigma) \text{ and } j\neq j_2(\sigma)} w_j C_j + w_{j_1(\sigma)}C_{j_2(\sigma)} + w_{j_2(\sigma)}C_{j_1(\sigma)} & \text{from 4.3} \\
& \leq & \sum_{j\neq j_1(\sigma) \text{ and } j\neq j_2(\sigma)} w_j C_j + w_{j_2(\sigma)}C_{j_2(\sigma)} + w_{j_1(\sigma)}C_{j_1(\sigma)} & \text{from 4.4} \\
& = & \sum w_j C_j &
\end{array}$$

Therefore, total weighted completion time does not increase as a result of the rescheduling. Since $j_1(\sigma') < j_1(\sigma)$, we need to apply the above rescheduling at most n times to obtain a schedule that satisfies the result stated in the Theorem. ■

4.2 A Classification of Problems

4.2.1 Sequence-Dependent Cases

The Open Shop Cases

Many of the fixed-sequence cases with sequence-dependent setup times are strongly $\mathcal{NP}$-hard. For example, the sequence-dependent open shop problem $1Om|s_{ii'}|C_{\max}$

is equivalent to the *traveling salesman problem.* (In a traveling salesman problem, a salesman has to visit each city exactly once and return to the original city in a tour that minimizes the total cost of visiting the cities. The cost of travelling from city i to i' is $a_{ii'}$.) The problem $1Om|s_{ii'}|C_{\max}$ is actually more closely related to the *Hamiltonian path problem,* which is an equivalent version of the traveling salesman problem. (A Hamiltonian path problem with a specified origin is a traveling salesman problem in which the salesman starts from the specified origin but does not return to it.) Given an instance of the problem $1Om|s_{ii'}|C_{\max}$ we can define an instance of the $(m+1)$-city Hamiltonian path problem in which the origin is 0 and the cost of travelling from city i to i' is $a_{ii'} = s_{ii'}\ \forall 0 \leq i \leq m,\ 1 \leq i' \leq m$. The total setup time of any schedule that sets up the machines in the order $(i_1, ..., i_m)$ is the same as the cost of visiting cities in the order $(0, i_1, ..., i_m)$. Hence, the problem $1Om|s_{ii'}|C_{\max}$ is equivalent to the Hamiltonian path problem with a specified origin. Garfinkel [43], in turn, shows the equivalence between the Hamiltonian path problem with a specified origin and the traveling salesman problem. Papadimitriou [91] shows that the traveling salesman problem is strongly $\mathcal{NP}$-hard even if the inter-city distances satisfy the triangle inequality conditions. Hence, the problem $1Om|s_{ii'}|C_{\max}$ is strongly $\mathcal{NP}$-hard even if the setup times satisfy the triangle inequality conditions.

Consider any regular objective $\delta \in \{L_{\max},\ \sum U_j,\ \sum w_j U_j,\ \sum C_j,\ \sum w_j C_j\}$. A single-job problem of the type $1Om|s_{ii'}|\delta$ is equivalent to the problem $1Om|s_{ii'}|C_{\max}$.

Hence, the problem $1Om|s_{ii'}|\delta$ is strongly $\mathcal{NP}$-hard even with a single job. Since the single-job cases are strongly $\mathcal{NP}$-hard, the fixed-sequence cases are strongly $\mathcal{NP}$-hard too.

The Flow Shop Cases

While the open shop problem $1Om|s_{ii'}|C_{\max}$ is strongly $\mathcal{NP}$-hard, the flow shop problem $1Fm|s_{ii'}|C_{\max}$ is solvable using a simple rule: *process all the jobs on each machine in any order in a single setup.* This rule will henceforth be called **Rule 1**.

The complexity status of problems $1Fm|s_{ii'}$, *fixed sequence*$|\sum w_jC_j$ and $1Fm|s_{ii'}$, *fixed sequence*$|L_{\max}$ is open. If the objective is the weighted number of tardy jobs, it is possible to address the integrated problem of sequencing and batching. Hence, we do not study the problem $1Fm|s_{ii'}$, *fixed sequence*$|\sum w_jU_j$. However, the problem $1Fm|s_{ii'}|\sum w_jU_j$ is $\mathcal{NP}$-hard even for $w_j = 1$ and $m = 2$. (This will follow from Theorem 6.1.)

4.2.2 Sequence-Independent Cases

Each of the problems $1Om|s_i|C_{\max}$ and $1Fm|s_i|C_{\max}$ is solvable using Rule 1. For the maximum lateness objective, the problem $1Om|s_i|L_{\max}$ turns out to be equivalent to a special case (Gerodimos et al. [46], Potts and Kovalyov [95]) of the problem $1|s_i|L_{\max}$. Recall that the problem $1|s_i|L_{\max}$ is an item-availability family scheduling problem in which there are F families of operations and a sequence-independent setup

time, s_i is required whenever a batch of operations belonging family i starts. Each job has a single operation and belongs to one of the F families.

The special case of the problem $1|s_i|L_{\max}$ which is equivalent to the problem $1Om|s_i|L_{\max}$ is of the following type: for each $i = 1, ..., m$ and $j = 1, ..., n$ family i has a job $(j-1)m + i$ with due date d_j. For example, if $m = 4$ and $n = 10$, family 1 comprises jobs $1, 5, 9, ..., 37$, family 2 comprises jobs $2, 6, 10, ..., 38$ and so on. Jobs 1, 2, 3 and 4 have due date d_1, jobs 5, 6, 7 and 8 have due date d_2 and so on. If we regard jobs 1, 2, 3 and 4 as separate operations of a fictitious single job with due date d_1, then the completion time of this fictitious job is the time when the last processed job among jobs 1, 2, 3 and 4 is completed. Hence, the lateness of such a fictitious job is the same as the maximum lateness of jobs 1, 2, 3 and 4.

Given an instance $\mathcal{I}_1$ of the problem $1Om|s_i|L_{\max}$ an instance $\mathcal{I}_2$ of the problem $1|s_i|L_{\max}$ with m families and setup times s_i is constructed as follows: for every operation (i, j) define a job $(j-1)m + i$ in family i with processing time t_{ij} and due date d_j. Consider a schedule σ_1 for instance $\mathcal{I}_1$ and a schedule σ_2 for instance $\mathcal{I}_2$ such that for two operations (i, j) and (i', j') the operation (i, j) precedes operation (i', j') in schedule σ_1 if and only if job $(j-1)m + i$ in family i precedes job $(j'-1)m + i'$ in family i'. Then, the lateness of jobs j in schedule σ_1 is the same as the maximum lateness of jobs $(j-1)m + 1$, ..., jm in schedule σ_2. Thus, maximum lateness is the same in schedules σ_1 and σ_2. This shows that the problem $1Om|s_i|L_{\max}$ is equivalent

to the above special case of the problem $1|s_i|L_{\max}$. Bruno and Downey [25] show that the above special case of the problem $1|s_i|L_{\max}$ is $\mathcal{NP}$-hard. Hence, the problem $1Om|s_i|L_{\max}$ is $\mathcal{NP}$-hard.

The complexity status of the flow shop problem $1Fm|s_i|L_{\max}$ is open. Problems $1Om|s_i|\sum w_jU_j$ and $1Fm|s_i|\sum w_jU_j$ are $\mathcal{NP}$-hard even for $w_j = 1$ and $m = 2$. (This will follow from Theorem 6.1). In Chapter 6 we shall show that both problems $1O2|s_i|\sum w_jU_j$ and $1F2|s_i|\sum w_jU_j$ are pseudo-polynomially solvable. The problem $1Fm|s_i$, *fixed sequence*$|\sum w_jC_j$ is polynomially solvable (see Section 7.2.1). The problem $1Om|s_i$, *fixed sequence*$|\sum w_jC_j$ is open.

The results are summarized in Table 4.1.

4.3 A Property of Some Objective Functions

Ghosh and Gupta [48] present a dynamic programming algorithm for the problem $1|s_{ii'}|L_{\max}$. As we have discussed in Section 4.2.2 the problem $1Om|s_{ii'}|L_{\max}$ can be reduced to a problem $1|s_{ii'}|L_{\max}$ with m families. Since the algorithm of Ghosh and Gupta has time complexity $O(m^2n^m)$ for m families, such a reduction gives an $O(m^2n^m)$ time algorithm for the problem $1Om|s_{ii'}|L_{\max}$.

However, in this chapter we shall develop a common dynamic programming scheme that applies to many fixed-sequence cases we study. To do this, we shall use the dynamic programming approach of Psaraftis [97], Ahn and Hyun [3], Ghosh [47] and

Problem	**Result**
$1Om\|s_{ii'}\|C_{\max}$	Strongly $\mathcal{NP}$-hard
$1Om\|s_i\|C_{\max}$	$O(1)$
$1Fm\|s_{ii'}\|C_{\max}$	$O(1)$
$1Fm\|s_i\|C_{\max}$	$O(1)$
$1Om\|s_{ii'}\|L_{\max}$	Strongly $\mathcal{NP}$-hard
$1Om\|s_i\|L_{\max}$	$\mathcal{NP}$-hard, not known if Strongly $\mathcal{NP}$-hard
$1Fm\|s_{ii'}\|L_{\max}$	Open
$1Fm\|s_i\|L_{\max}$	Open
$1Om\|s_{ii'}\|\sum w_jU_j$	Strongly $\mathcal{NP}$-hard even if $w_j = 1$
$1Om\|s_i\|\sum w_jU_j$	$\mathcal{NP}$-hard even if $w_j = 1$, pseudo-polynomially solvable if $m = 2$ (see Section 6.2)
$1Fm\|s_{ii'}\|\sum w_jU_j$	$\mathcal{NP}$-hard even if $w_j = 1$, pseudo-polynomially solvable if $m = 2$ (see Section 6.3)
$1Fm\|s_i\|\sum w_jU_j$	$\mathcal{NP}$-hard even if $w_j = 1$, pseudo-polynomially solvable if $m = 2$ (see Section 6.3)
$1Om\|s_{ii'}$, *fixed sequence*$\|\sum w_jC_j$	Strongly $\mathcal{NP}$-hard even if $w_j = 1$
$1Om\|s_i$, *fixed sequence*$\|\sum w_jC_j$	Open
$1Fm\|s_{ii'}$, *fixed sequence*$\|\sum w_jC_j$	Open
$1Fm\|s_i$, *fixed sequence*$\|\sum w_jC_j$	$O(mn^3)$ (see Section 7.2.1)

Table 4.1: Classification of one-operator problems

Ghosh and Gupta [48] and apply the dominance properties discussed in Chapter 3.

First, we shall discuss a property of some objective functions which is important for getting a common scheme for various objectives. Each of makespan, maximum lateness and total (weighted) completion time is obtained from some linear functions of completion times. For $\Delta \in \{C_{\max}, L_{\max}, \sum C_j, \sum w_j C_j\}$ the objective Δ is of the type $\max\{\Delta_j(C_j)\}$ or $\sum \Delta_j(C_j)$, where $\Delta_j(C_j) = \varepsilon_j + \nu_j C_j$.

- For makespan, $\varepsilon_j = 0$, $\nu_j = 1$ and $\Delta = \max\{\Delta_j(C_j)\}$.
- For maximum lateness, $\varepsilon_j = -d_j$, $\nu_j = 1$, and $\Delta = \max\{\Delta_j(C_j)\}$.
- For total (weighted) completion time, $\varepsilon_j = 0$, $\nu_j = w_j$ and $\Delta = \sum \Delta_j(C_j)$.

Note that the total (weighted) number of tardy jobs is of the type $\sum \Delta_j(C_j)$, where each Δ_j is a nonlinear function of completion time C_j. The nonlinearity of the Δ_j in this case implies that a different scheme must be used to solve the problem.

Each objective $\Delta \in \{C_{\max}, L_{\max}, \sum w_j C_j\}$ possesses an interesting property. Let $\Delta(\sigma)$ be the objective function value given by the schedule σ and let schedule $\sigma(t')$ be the schedule obtained from σ by delaying the start and completion time of every operation by t' time units. The following result holds:

Remark 4.1 *For any objective $\Delta \in \{C_{\max}, L_{\max}, \sum C_j, \sum w_j C_j\}$, if σ^* is an optimal schedule when the processing starts at time t, then $\sigma^*(t')$ is an optimal schedule when*

the processing starts at time $t + t'$. *Furthermore,* $\Delta(\sigma^*(t')) = \Delta(\sigma^*) + t'$ *if* $\Delta \in \{C_{\max}, L_{\max}\}$ *and* $\Delta(\sigma^*(t')) = \Delta(\sigma^*) + t' \sum w_j$ *if* $\Delta = \sum w_j C_j$.

Proof: Consider any schedule σ that starts at time t. Let C_j be the completion time of job j in schedule σ . If $\Delta \in \{C_{\max}, L_{\max}\}$, then $\Delta(\sigma(t')) = \max\{\Delta_j(C_j + t')\} = \max\{\varepsilon_j + C_j + t'\} = \max\{\varepsilon_j + C_j\} + t' = \Delta(\sigma) + t'$. If $\Delta = \sum w_j C_j$, then $\Delta(\sigma(t')) = \sum \Delta_j(C_j + t') = \sum w_j(C_j + t') = \sum w_j C_j + t' \sum w_j = \Delta(\sigma) + \sum w_j t'$. Hence, $\min\{\Delta(\sigma)\} = \Delta(\sigma^*) \Rightarrow \min\{\Delta(\sigma(t'))\} = \Delta(\sigma^*(t'))$. ■

Thus, the optimal sequence for the objectives makespan, maximum lateness and weighted completion time is *independent of the start time* of the schedule. If a sequence is optimal for one start time, then the sequence is optimal for all other start times.

It is natural to ask at this point whether for every objective of the type $\max\{\Delta_j(C_j)\}$ or $\sum \Delta_j(C_j)$ the optimal sequence is independent of the start time of the schedule if $\Delta_j(C_j) = \varepsilon_j + \nu_j C_j$. A slight modification of the above proof shows that the answer is 'yes' for the objectives of the type $\sum \Delta_j(C_j)$. However, the answer is 'no' for at least one objective of the type $\max\{\Delta_j(C_j)\}$. We can think of an objective that may be called maximum weighted lateness: $\delta = \max\{w_j L_j\}$. In this case, $\varepsilon_j = -d_j$ and $\nu_j = w_j$. For this objective an optimal sequence depends on the start time of the schedule. To see this, consider the following example.

Example 4.1 *Consider an instance of the classical single machine problem with the*

following processing times, due dates and weights:

j	1	2
t_j	1	1
d_j	1	2
w_j	1	3

Suppose that the schedule starts at time zero. If job 1 is processed before job 2, then $C_1 = 1$, $C_2 = 2$ and $\max\{w_jL_j\} = 0$. If job 2 is processed before job 1, then $C_1 = 2$, $C_2 = 1$ and $\max\{w_jL_j\} = 1$. Hence, $\max\{w_jL_j\}$ is minimized by processing job 1 before job 2. Now, suppose that the schedule starts at time 1. If job 1 is processed before job 2, then $C_1 = 2$, $C_2 = 3$ and $\max\{w_jL_j\} = 3$. If job 2 is processed before job 1, then $C_1 = 3$, $C_2 = 2$ and $\max\{w_jL_j\} = 2$. Hence, $\max\{w_jL_j\}$ is minimized by processing job 2 before job 1. Thus, an optimal sequence for the objective $\max\{w_jL_j\}$ is not independent of the start time of the schedule.

Since an optimal sequence for $\Delta \in \{C_{\max}, L_{\max}, \sum w_jC_j\}$ is independent of the start time of the schedule, we can apply the approach of Psaraftis [97], Ahn and Hyun [3], Ghosh [47] and Ghosh and Gupta [48] and obtain an algorithm for $1Om|s_{ii'}$,*fixed sequence*$|\Delta$ and $1Fm|s_i$,*fixed sequence*$|\Delta$. However, as the flow shop case with the makespan objective is solved using Rule 1, we do not discuss the problem $1Fm|s_i$,*fixed sequence*$|C_{\max}$.

4.4 A Dynamic Program for some Fixed-Sequence Cases

Throughout this section we assume that the job-orders on all m machines are the same and known. We may thus assume, by relabelling if necessary, that the jobs are processed in the order 1, 2, ..., n. Suppose that on each machine i the operator has processed the first n_i jobs. For the flow shop case, $n \geq n_1 \geq n_2 \geq ... \geq n_m \geq 0$. For the open shop case $0 \leq n_i \leq n$ for each i. The remaining operations are $\{(i, n_i + 1), (i, n_i + 2), ..., (i, n) : 1 \leq i \leq m\}$. Suppose that machine i^* is the current machine. The operator can restrict the choice of the next activity to one of the following:

1. process operation $(i^*, n_{i^*} + 1)$; or

2. set up a machine $i^0 \neq i^*$.

We shall now point out some cases when the operator can further restrict the choice of the next activity. One such case is $n_{i^*} = n$, when there is no operation $(i^*, n_{i^*} + 1)$ left to process. In such a case, if $\min\{n_{i'}\} < n$, one of the machines $i^0 \neq i^*$ is set up and if $\min\{n_{i'}\} = n$, all the operations are complete and the process terminates. Further restrictions are as follows:

1. **Restriction on setup:** The operator does not switch from the current machine i^* to a machine i^0 if machine i^* has no inventory unless, however, the current machine $i^* = m$ and the machine environment is a flow shop. In the flow shop

environment the operator continues working on machine m as long as there is a job waiting for machine m. Hence, we get the following restrictions on the choices of the operator:

(a) In the open shop case, if $\min\{n_{i'}\} = n_{i^*} < n$, then machine i^* does not have a positive inventory and, therefore, the operator processes operation $(i^*, n_{i^*} +1)$. This case is analyzed separately depending on whether job $(n_{i^*} + 1)$ is completed with the completion of operation $(i^*, n_{i^*} +1)$. If $n_{i^*} < n_i$ for all $i \neq i^*$, then job $(n_{i^*} + 1)$ is completed with the completion of operation $(i^*, n_{i^*} +1)$. If there exists at least one $i \neq i^*$ with $n_i = n_{i^*}$, then job $(n_{i^*} +1)$ is not completed with the completion of operation $(i^*, n_{i^*} +1)$.

(b) In the flow shop case, if $i^* = m$ and $n_m < n_i \leq n$ for all $i \neq m$, then every machine $i \neq m$ has a positive inventory and, therefore, the operator processes operation $(m, n_m +1)$ and job $(n_m +1)$ is completed with the completion of operation $(i^*, n_{i^*} +1)$.

(c) In the flow shop case, if $i^* < m$ and $n_{i^*} = n_m < n$, then machine i^* does not have a positive inventory and, therefore, the operator processes operation $(i^*, n_{i^*} +1)$. Job $(n_m +1)$ is not completed with the completion of operation $(i^*, n_{i^*} +1)$.

2. **Restriction on the choice of machines for setup:** A machine is not set up as long as it has a positive inventory. Therefore, machine i^0 is a candidate for setup only if $n_{i^0} \leq n_i$ for all i. Further restrictions on the choice of machine i^0 are as follows:

   (a) In the open shop case, if the operator switches from the current machine i^* to a machine i^0, then: (i) machine i^* has a positive inventory; and (ii) machine i^0 has no inventory. From condition (i), $\min\{n_{i'}\} < n_{i^*}$, so $n_{i^0} < n_{i^*}$. From condition (ii), $n_{i^0} \leq n_i$ for all i.

   (b) In the flow shop case, if the current machine $i^* = m$, as soon as all the jobs waiting for machine m are processed, the operator stops working on machine m and switches to the least indexed machine i^0 with no inventory. Hence, $i^0 = \min\{i : n_i = n_m\} \neq m$.

   (c) In the flow shop case, if the current machine $i^* < m$, then each machine i_1, $1 \leq i_1 < i^*$ has a positive inventory and each machine i_2, $1 \leq i_2 < i^*$ has zero inventory. The operator switches from machine i^* to the next machine $i^* + 1$ after building a positive inventory on machine i^*. Hence, $n_{i^*+1} < n_{i^*}$.

We are now ready to describe a backward dynamic programming scheme for $1Om|s_{ii'}$,*fixed sequence*$|\Delta$, where $\Delta \in \{C_{\max}, L_{\max}, \sum w_j C_j\}$ and $1Fm|s_i$,*fixed se-*

quence$|\Delta$, where $\Delta \in \{L_{\max}, \sum w_j C_j\}$. At each stage of the dynamic program we optimally schedule operations $\{(i, n_i+1), (i, n_i+2), ..., (i, n) : 1 \leq i \leq m\}$ at time zero.

Let $h(n_1, n_2, ..., n_m, i^*)$ = minimum value of objective $\Delta \in \{C_{\max}, L_{\max}, \sum w_j C_j\}$ over all partial schedules that processes only operations $\{(i, n_i+1), (i, n_i+2), ..., (i, n) : 1 \leq i \leq m\}$ starting from time zero, given that machine i^* is the current machine at time zero. In the following we define $W_j = \sum_{j'=j}^{n} w_{j'}$ for $j = 1, 2, ..., n$.

4.4.1 Open Shop Sequence-Dependent Cases

We shall now discuss the computation of $h(n_1, n_2, ..., n_m, i^*), 0 \leq n_i \leq n, 1 \leq i \leq m, 1 \leq i^* \leq m$ for $1Om|s_{ii'}$,*fixed sequence*$|\Delta$, where $\Delta \in \{C_{\max}, L_{\max}, \sum w_j C_j\}$. Suppose that the following values are computed before computing $h(n_1, n_2, ..., n_m, i^*)$:

$h(n'_1, n'_2, ..., n'_m, i^*)$, $n'_i = n_i$, if $i \neq i^*$ and $n'_{i^*} = n_{i^*} + 1$ and
$h(n'_1, n'_2, ..., n'_m, i^0)$, $n'_i = n_i, \forall i$ and $n_{i^0} = \min\{n_{i'}\} < n_{i^*}$.

Define:

$$\begin{aligned} h &= h(n_1, n_2, ..., n_m, i^*) \\ h_{i^*} &= h(n'_1, n'_2, ..., n'_m, i^*), \quad n'_i = n_i, \text{if } i \neq i^* \text{ and } n'_{i^*} = n_{i^*} + 1 \text{ and} \\ h_{i^0} &= h(n'_1, n'_2, ..., n'_m, i^0), \quad n'_i = n_i, \forall i \text{ and } n_{i^0} = \min\{n_{i'}\} < n_{i^*}. \end{aligned}$$

In the following we shall compute h. We shall assume that only operations $\{(i, n_i+1), (i, n_i+2), ..., (i, n) : 1 \leq i \leq m\}$ are to be processed and the processing of operations $\{(i, n_i+1), (i, n_i+2), ..., (i, n) : 1 \leq i \leq m\}$ starts at time zero when machine i^* is the current machine. The operator has two choices: either process operation $(i^*, n_{i^*}+1)$, or set up a machine.

Case 1: $\min\{n_{i'}\} = n_{i^*} < n$. The current machine i^* does not have a positive inventory. There is only one choice for the operator, namely to process operation $(i^*, n_{i^*}+1)$. If $\Delta = C_{\max}$, we get

$$h = h_{i^*} + t_{i^*,n_{i^*}+1}.$$

To get an expression for h when $\Delta \in \{L_{\max}, \sum w_j C_j\}$ we consider two subcases. In one subcase the operation $(i^*, n_{i^*}+1)$ is the last operation of job $(n_{i^*}+1)$, while in the other, it is not. In both cases operation $(i^*, n_{i^*}+1)$ precedes completion of each job j, $\min\{n_{i'}\}+1 \leq j \leq n$.

Case 1a: $\min\{n_{i'}\} = n_{i^*} < n_i \leq n$ for all $i \neq i^*$. Operation $(i^*, n_{i^*}+1)$ is the last operation of job $(n_{i^*}+1)$ and, therefore, the job $(n_{i^*}+1)$ is complete as soon as operation $(i^*, n_{i^*}+1)$ is complete. We get

$$h = \begin{cases} \max\{t_{i^*,n_{i^*}+1} - d_{n_{i^*}+1}, h_{i^*} + t_{i^*,n_{i^*}+1}\} & \text{if } \Delta = L_{\max} \\ h_{i^*} + t_{i^*,n_{i^*}+1} W_{n_{i^*}+1} & \text{if } \Delta = \sum w_j C_j. \end{cases}$$

Case 1b: $\min\{n_{i'}\} = n_{i^*} = n_i < n$ for at least one $i \neq i^*$. Operation $(i^*, n_{i^*}+1)$ is not the last operation of job $(n_{i^*}+1)$. Job $(n_{i^*}+1)$ has an unprocessed operation at least on machine i. Thus

$$h = \begin{cases} h_{i^*} + t_{i^*,n_{i^*}+1} & \text{if } \Delta = L_{\max} \\ h_{i^*} + t_{i^*,n_{i^*}+1} W_{n_{i^*}+1} & \text{if } \Delta = \sum w_j C_j. \end{cases}$$

Case 2: $\min\{n_{i'}\} < n_{i^*} < n$. For the makespan objective this case does not occur because we assume $n = 1$. Hence, consider $\Delta \in \{L_{\max}, \sum w_j C_j\}$. In this case,

one choice of the operator is to process operation $(i^*, n_{i^*}+1)$. Operation $(i^*, n_{i^*}+1)$ cannot be the last operation of job $(n_{i^*}+1)$ because on each machine i^0 with $n_{i^0} = \min\{n_{i'}\}$, job $(n_{i^*}+1)$ has an unprocessed operation. Another choice of the operator is to set up any machine i^0 with $n_{i^0} = \min\{n_{i'}\}$. Thus

$$h = \begin{cases} \min\{h_{i^*} + t_{i^*,n_{i^*}+1}, h_{i^0} + s_{i^*i^0} : n_{i^0} = \min\{n_{i'}\}\} & \text{if } \Delta = L_{\max} \\ \min\{h_{i^*} + t_{i^*,n_{i^*}+1} W_{\min\{n_{i'}\}+1}, \\ h_{i^0} + s_{i^*i^0} W_{\min\{n_{i'}\}+1} : n_{i^0} = \min\{n_{i'}\}\} & \text{if } \Delta = \sum w_j C_j. \end{cases}$$

Case 3: $\min\{n_{i'}\} < n_{i^*} = n$. All the operations on the current machine i^* are processed. The operator sets up one of the machines i^0 with $n_{i^0} = \min\{n_{i'}\}$.

$$h = \begin{cases} \min\{h_{i^0} + s_{i^*i^0} : n_{i^0} = \min\{n_{i'}\}\} & \text{if } \Delta \in \{C_{\max}, L_{\max}\} \\ \min\{h_{i^0} + s_{i^*i^0} W_{\min\{n_{i'}\}+1} : n_{i^0} = \min\{n_{i'}\}\} & \text{if } \Delta = \sum w_j C_j. \end{cases}$$

Initialization and optimal value: For all i^* and $n_1 = n_2 = ... = n_m = n$ the objective function value $h(n_1, n_2, ..., n_m, i^*)$ is initialized as follows:

$$h(n_1, n_2, ..., n_m, i^*) = \begin{cases} 0 & \text{if } \Delta = C_{\max} \\ -\infty & \text{if } \Delta = L_{\max} \\ 0 & \text{if } \Delta = \sum w_j C_j \end{cases}$$

The optimal objective value is given as

$$h^* = \begin{cases} \min\{h(n_1, n_2, ..., n_m, i^0) \\ +s_{0i^0} : 1 \le i^0 \le m\}, \quad n_i = 0 \forall i & \text{if } \Delta \in \{C_{\max}, L_{\max}\} \\ \min\{h(n_1, n_2, ..., n_m, i^0) \\ +s_{0i^0} W_1 : 1 \le i^0 \le m\}, \quad n_i = 0 \forall i & \text{if } \Delta = \sum w_j C_j \end{cases}$$

Running time: The number of $h(n_1, n_2, ..., n_m, i^*)$ values computed is, $O(m(n+1)^m)$ and each value $h(n_1, n_2, ..., n_m, i^*)$ is computed in $O(m)$ time. Hence, the running time of the above dynamic programming scheme is $O(m^2(n+1)^m)$. However, in case of $\Delta = C_{\max}$, we may assume that $n = 1$. Hence, the running time is $O(m^2 2^m)$.

Theorem 4.3 *Problem* $1Om|s_{ii'}$,fixed sequence$|\Delta$, *is solved in* $O(m^2(n+1)^m)$ *time if* $\Delta \in \{L_{\max}, \sum w_jC_j\}$ *and in* $O(m^2 2^m)$ *time if* $\Delta = C_{\max}$.

Corollary 4.3.1 *The problem* $1Om|s_{ii'}|C_{\max}$ *is solved in* $O(m^2 2^m)$ *time.*

Corollary 4.3.2 *The problem* $1Om|s_{ii'}|L_{\max}$ *is solved in* $O(m^2(n+1)^m)$ *time.*

Corollary 4.3.3 *The problem* $1Om|s_{ii'}$,aptw$|\sum w_jC_j$ *is solved in* $O(m^2(n+1)^m)$ *time.*

4.4.2 Flow Shop Sequence-Independent Cases

We shall now discuss the computation of $h(n_1, n_2, ..., n_m, i^*)$, $n \geq n_1 \geq n_2 \geq ... \geq n_m \geq 0$, $1 \leq i^* \leq m$ for $1Fm|s_i$,*fixed sequence*$|\Delta$, where $\Delta \in \{L_{\max}, \sum w_jC_j\}$. Suppose that the following values are computed before computing $h(n_1, n_2, ..., n_m, i^*)$:

$$\begin{array}{ll} h(n_1', n_2', ..., n_m', i^*), & n_i' = n_i, \text{ if } i \neq i^* \text{ and } n_{i^*}' = n_{i^*} + 1 \\ h(n_1', n_2', ..., n_m', i^* + 1), & n_i' = n_i, \forall i \\ h(n_1', n_2', ..., n_m', i^0), & n_i' = n_i, \forall i \text{ and } i^0 = \min\{i : n_i = n_m\}. \end{array}$$

Define:

$$\begin{array}{lcll} h & = & h(n_1, n_2, ..., n_m, i^*) & \\ h_{i^*} & = & h(n_1', n_2', ..., n_m', i^*), & n_i' = n_i, \text{ if } i \neq i^* \text{ and } n_{i^*}' = n_{i^*} + 1 \\ h_{i^*+1} & = & h(n_1', n_2', ..., n_m', i^* + 1), & n_i' = n_i, \forall i \\ h_{i^0} & = & h(n_1', n_2', ..., n_m', i^0), & n_i' = n_i, \forall i \text{ and } i^0 = \min\{i : n_i = n_m\}. \end{array}$$

In the following, we shall compute h. We shall assume that only operations $\{(i, n_i+1), (i, n_i+2), ..., (i, n) : 1 \leq i \leq m\}$ are to be processed and the processing of operations $\{(i, n_i+1), (i, n_i+2), ..., (i, n) : 1 \leq i \leq m\}$ starts at time zero when machine i^* is the current machine. The operator has two choices: either process operation $(i^*, n_{i^*}$

+1) or set up a machine. If $i^* < m$, only machine (i^* +1) is a candidate for setup. If $i^* = m$, only machine i^0 is a candidate for setup, where $i^0 = \min\{i : n_i = n_m\}$.

Case 1: $n_{i^*} = n_m < n$. The current machine i^* does not have a positive inventory. We analyze this case in three separate subcases. In each subcase the operator has only one choice of the next activity which precedes completion of each job j, (n_m +1) $\leq j \leq n$.

Case 1a: $i^* = m$ and $n_m < n_i \leq n$ for all $i \neq m$. The operator processes operation (i^*, n_{i^*} +1) which is the last operation of job (n_{i^*} +1). Thus

$$h = \begin{cases} \max\{t_{m,n_m+1} - d_{n_m+1}, h_m + t_{m,n_m+1}\} & \text{if } \Delta = L_{\max} \\ h_m + t_{m,n_m+1} W_{n_m+1} & \text{if } \Delta = \sum w_j C_j. \end{cases}$$

Case 1b: $i^* = m$ and $n_m = n_i < n$ for at least one $i \neq i^*$. The operator sets up machine i^0 such that $i^0 = \min\{i : n_i = n_m\}$. Thus

$$h = \begin{cases} h_{i^0} + s_{i^0} & \text{if } \Delta = L_{\max} \\ h_{i^0} + s_{i^0} W_{n_m+1} & \text{if } \Delta = \sum w_j C_j. \end{cases}$$

Case 1c: $i^* < m$ and $n_{i^*} = n_m < n$. The operator processes operation (i^*, n_{i^*} +1) which is not the last operation of job (n_{i^*} +1). Thus

$$h = \begin{cases} h_{i^*} + t_{i^*,n_{i^*}+1} & \text{if } \Delta = L_{\max} \\ h_{i^*} + t_{i^*,n_{i^*}+1} W_{n_m+1} & \text{if } \Delta = \sum w_j C_j. \end{cases}$$

Case 2: $i^* < m$ and $n_m < n_{i^*} < n$. One choice of the operator is to process operation (i^*, n_{i^*} +1). In this case, operation (i^*, n_{i^*} +1) cannot be the last operation of job (n_{i^*} +1) because $i^* \neq m$. Another choice of the operator is to set up the next machine (i^* + 1). Thus

$$h = \begin{cases} \min\{h_{i^*} + t_{i^*,n_{i^*}+1}, h_{i^*+1} + s_{i^*+1}\} & \text{if } \Delta = L_{\max} \\ \min\{h_{i^*} + t_{i^*,n_{i^*}+1} W_{n_m+1}, h_{i^*+1} + s_{i^*+1} W_{n_m+1}\} & \text{if } \Delta = \sum w_j C_j. \end{cases}$$

Case 3: $i^* < m$ and $n_m < n_{i^*} = n$. The operator sets up the next machine $(i^* + 1)$.

Thus

$$h = \begin{cases} h_{i^*+1} + s_{i^*+1} & \text{if } \Delta = L_{\max} \\ h_{i^*+1} + s_{i^*+1} W_{n_m+1} & \text{if } \Delta = \sum w_j C_j. \end{cases}$$

Initialization and optimal value: For $n_1 = n_2 = ... = n_m = n$ the objective function value $h(n_1, n_2, ..., n_m, m)$ is initialized as follows:

$$h(n_1, n_2, ..., n_m, m) = \begin{cases} -\infty & \text{if } \Delta = L_{\max} \\ 0 & \text{if } \Delta = \sum w_j C_j \end{cases}$$

The optimal objective value is given as

$$h^* = \begin{cases} h(n_1, n_2, ..., n_m, 1) + s_1, & n_i = 0 \forall i \quad \text{if } \Delta = L_{\max} \\ h(n_1, n_2, ..., n_m, 1) + s_1 W_1, & n_i = 0 \forall i \quad \text{if } \Delta = \sum w_j C_j \end{cases}$$

Running time: The running time is slightly better in the flow shop sequence-independent case than open shop sequence-dependent case. The number of $h(n_1, n_2, ..., n_m, i^*)$ values computed is $O(m(n+1)^m)$, and each value $h(n_1, n_2, ..., n_m, i^*)$ is computed in constant time. Hence, the running time of the above dynamic programming scheme is $O(m(n+1)^m)$.

Theorem 4.4 *The problem* $1Fm|s_i$,fixed sequence$|\Delta$, $\Delta \in \{L_{\max}, \sum w_j C_j\}$ *is solved in time* $O(m(n+1)^m)$.

Corollary 4.4.1 *The problem* $1Fm|s_i|L_{\max}$ *is solved in* $O(m(n+1)^m)$ *time.*

Corollary 4.4.2 *The problem* $1Fm|s_i$,aptw$|\sum w_jC_j$ *is solved in* $O(m(n+1)^m)$ *time.*

We must note here that the problem $1Fm|s_i$,*fixed sequence*$|\sum w_jC_j$ and, therefore, the problem $1Fm|s_i, aptw|\sum w_jC_j$ can be solved more efficiently. This will be shown in Chapter 7.

4.5 Summary

In this Chapter we consider the fixed-sequence case, in which the job-orders on all the machines are known and fixed. Theorems 4.1 and 4.2 show the importance of the fixed-sequence cases. For minimizing maximum lateness, there exists an optimal schedule in which the jobs are processed in the EDD order on all machines. For minimizing total weighted completion time, there exists an optimal schedule in which jobs are processed in the order 1, 2,..., n on all machines, if the processing times and weights are agreeable. Hence, a solution to the fixed-sequence case, in turn solves a large number of cases.

We use the dominance properties developed in Chapter 3 and apply the dynamic programming approach of Psaraftis [97], Ahn and Hyun [3], Ghosh [47] and Ghosh and Gupta [48] in order to solve the problems $1Om|s_{ii'}$,*fixed sequence*$|\Delta$ for $\Delta \in \{C_{\max}, L_{\max}, \sum w_jC_j\}$ and $1Fm|s_i$,*fixed sequence*$|\Delta$ for $\Delta \in \{L_{\max}, \sum w_jC_j\}$.

In Table 4.2, we summarize the running times of the dynamic program for various fixed-sequence cases and compare them with those of some existing algorithms. Note

that we improve the running time of the problem $1Fm|s_i,aptw|\sum w_jC_j$ to $O(mn^3)$ in Chapter 7.

Problem	**Result (Chapter 4)**	**Previously Known Result**
$1Om\|s_i\|L_{\max}$	$O(m^2(n+1)^m)$	$O(m^2(n+1)^m)$ (Ghosh and Gupta [48], Gerodimos et al. [46])
$1Om\|s_i,aptw\|\sum w_jC_j$	$O(m^2(n+1)^m)$	$O(m^2(n+1)^m)$ (Gerodimos et al. [46])
$1Fm\|s_i\|L_{\max}$	$O(m^2(n+1)^m)$	—
$1Fm\|s_i,aptw\|\sum w_jC_j$	$O(m^2(n+1)^m)$	—
$1Om\|s_{ii'}\|C_{\max}$	$O(m^2 2^m)$	—
$1Om\|s_{ii'}\|L_{\max}$	$O(m^2(n+1)^m)$	$O(m^2(n+1)^m)$ (Ghosh and Gupta [48])
$1Om\|s_{ii'},aptw\|\sum w_jC_j$	$O(m^2(n+1)^m)$	—

Table 4.2: Running times of some fixed-sequence cases

Chapter 5

Two-Machine Problem with Maximum Lateness Objective

In this chapter we shall consider problems $1O2|s_i|L_{\max}$ and $1F2|s_i|L_{\max}$. The dynamic programming scheme discussed in Chapter 4 can be used to solve both problems in $O(n^2)$ time. The problem $1O2|s_i|L_{\max}$ can be solved in $O(n^2)$ time by a dynamic programming recursion given by Gerodimos et al. [45]. Yet another strategy (Gerodimos et al. [46], Potts and Kovalyov [95]) to solve the problem $1O2|s_i|L_{\max}$ is to apply the algorithm of Ghosh and Gupta [48] directly by reducing the problem $1O2|s_i|L_{\max}$ to the special case of the problem $1|s_{ii'}|L_{\max}$ as discussed in Section 4.2.2. The two-machine problem $1O2|s_i|L_{\max}$ reduces to a two-family problem. The algorithm of Ghosh and Gupta [48] has a running time $O(m^2n^m)$ for m families. Hence, the problem $1O2|s_i|L_{\max}$ is solved in $O(n^2)$ time.

It is not known whether the flow shop problem $1F2|s_i|L_{\max}$ can be reduced to the problem $1|s_{ii'}|L_{\max}$. However, another previously studied problem is related to the

problem $1F2|s_i|L_{\max}$. As we have discussed in Section 3.5, the problem $1|s_1, F = 1|\delta$ is a special case of the problem $1F2|s_i|\delta$. Webster and Baker [121] discuss an $O(n^2)$ time algorithm for the problem $1|s_1, F = 1|L_{\max}$.

In this chapter we shall show that by using a common dynamic programming scheme, both the problems $1O2|s_i|L_{\max}$ and $1F2|s_i|L_{\max}$ can be solved in $O(n)$ time after due date sorting. We thus obtain a common approach which solves both the open shop and flow shop cases. Moreover, we obtain an improvement on the algorithm of Webster and Baker [121] for the problem $1|s_1, F = 1|L_{\max}$.

It follows from Theorems 3.6 and 4.1 that there exists an optimal schedule which is a batching schedule and in which the jobs are arranged in the Earliest Due Date (EDD) order. Hence, we may assume, by relabelling if necessary that the jobs are arranged in the order 1, 2, ..., n. Our problem thus reduces to identifying the batches and, in the open shop case, deciding which machine to start first.

5.1 The Open Shop Problem

Let τ_{ij} denote the total processing time of batch $[i, j]$: $\tau_{ij} = \sum_{l=i}^{j}(t_{1l} + t_{2l})$. Define the following quantities:

f_{ij} (resp., g_{ij}) is the maximum lateness of jobs in batch $[i, j]$ if post-setup processing of this batch starts at time zero on machine 1 (resp., machine 2);

f^*_{ij} (resp., g^*_{ij}) is the optimal maximum lateness of jobs $i, \ldots, n$, given that batch $[i, j]$ starts post-setup processing at time zero on machine 1 (resp., machine 2); and

L_{1i} (resp., L_{2i}) is the optimal maximum lateness of jobs $i, \ldots, n$, given that job i starts post-setup processing at time zero on machine 1 (resp., machine 2).

The optimal value of maximum lateness is $L^*_{\max} = \min\{L_{11} + s_1, L_{21} + s_2\}$.

The following equations—which appear, superficially to involve circularity—can be implemented in a recursive, non-circular manner. For $1 \le i \le j \le n$:

$$f_{ij} = \max_{i \le k \le j}\{\sum_{l=i}^{k} t_{2l} - d_k\} + s_2 + \sum_{l=i}^{j} t_{1l} \tag{5.1}$$

$$g_{ij} = \max_{i \le k \le j}\{\sum_{l=i}^{k} t_{1l} - d_k\} + s_1 + \sum_{l=i}^{j} t_{2l} \tag{5.2}$$

$$f^*_{ij} = \begin{cases} f_{in} & \text{for } j = n \\ \max\{f_{ij}, L_{2(j+1)} + \tau_{ij} + s_2\} & \text{for } j < n \end{cases} \tag{5.3}$$

$$g^*_{ij} = \begin{cases} g_{in} & \text{for } j = n \\ \max\{g_{ij}, L_{1(j+1)} + \tau_{ij} + s_1\} & \text{for } j < n \end{cases} \tag{5.4}$$

$$L_{1i} = \min_{i \le j \le n}\{f^*_{ij}\} \tag{5.5}$$

$$L_{2i} = \min_{i \le j \le n}\{g^*_{ij}\}. \tag{5.6}$$

Equations (5.1)–(5.4) determine f_{nn}, g_{nn}, f^*_{nn} and g^*_{nn}. Then equations (5.5)–(5.6) determine L_{1n} and L_{2n}, from which equations (5.1)–(5.4) determine f_{ij}, g_{ij}, f^*_{ij} and g^*_{ij} for $i = n-1$ and $j = n-1, n$. Continuing in this way determines the quantities in (5.3)–(5.6) for all i and j. A straightforward implementation of this type will

determine an optimal batching schedule in $O(n^2)$ computations. However, we can reduce the computational complexity to $O(n)$ through a more detailed analysis.

5.1.1 A Network Representation

Before we move on, let us discuss a network representation of the problem. Introduce a dummy job $(n+1)$. Define a directed network $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with node-set $\mathbf{V} = \{0\} \cup \mathbf{V}_1 \cup \mathbf{V}_2$ and arc-set $\mathbf{E} = \mathbf{E}_0 \cup \mathbf{E}_1 \cup \mathbf{E}_2$. For each operation $(1, j)$, there is a node a_j in $\mathbf{V}_1$. For each operation $(2, j)$, there is a node b_j in $\mathbf{V}_2$. Arc-set $\mathbf{E}_0$ contains only two arcs $\langle 0, a_1 \rangle$ and $\langle 0, b_1 \rangle$. For all $1 \leq j < j' \leq (n+1)$ there is an arc $\langle a_j, b_{j'} \rangle$ in $\mathbf{E}_1$ and an arc $\langle b_j, a_{j'} \rangle$ in $\mathbf{E}_2$.

Node 0 represents the initial state. Arc $\langle 0, a_1 \rangle$ represents the setup of machine 1 at time zero. Arc $\langle 0, b_1 \rangle$ represents the setup of machine 2 at time zero. Each arc $\langle a_j, b_{j'} \rangle$ represents processing of batch $[j, j'-1]$ starting on machine 1 and each arc $\langle b_j, a_{j'} \rangle$ represents processing of batch $[j, j'-1]$ starting on machine 2.

Each arc e is associated with two weights $r(e)$ and $r'(e)$. The weight $r(e)$ represents a maximum lateness and weight $r'(e)$ represents total processing time. Consider an arc $e = \langle a_j, b_{j'} \rangle$. The weight $r(e) = f_{j(j'-1)}$ is the maximum lateness of jobs in batch $[j, j'-1]$ if the batch starts post-setup processing on machine 1 at time zero. The weight $r'(e) = \tau_{j(j'-1)} + s_2$ is the length of time by which the batch $[j, j'-1]$ delays all the operations and setups that follow batch $[j, j'-1]$. Similarly, for arc $e = \langle b_j, a_{j'} \rangle$, we have $r(e) = g_{j(j'-1)}$ and $r'(e) = \tau_{j(j'-1)} + s_1$. Arc $e = \langle 0, a_1 \rangle$ does not represent

any batch, so we set $r(e) = -\infty$. However, the arc represents a setup operation on machine 1, so we set $r'(e) = s_1$. Similarly, for arc $e = \langle 0, b_1 \rangle$ we have $r(e) = -\infty$ and $r'(e) = s_2$.

The length of a path $\pi = \langle e_1, e_2, ..., e_k \rangle$ is

$$L(\pi) = \max_{1 \le u \le k} \{ r(e_u) + \sum_{v=1}^{u-1} r'(e_v) \} \tag{5.7}$$

If path π starts from node 0, then π represents an initial setup and some batches. In this case, the length of π represents the maximum lateness when the initial setup and all the batches represented by π are carried out at time zero. If path π does not start from node 0, then π represents only some batches. In this case, the length of π represents the maximum lateness when all the batches represented by π are carried out at time zero.

The problem is to find a shortest path from node 0 to a_{n+1} or b_{n+1}.

Consider a path $\pi = \langle e_1, e_2, ..., e_k \rangle$ with $k > 1$ and its subpath $\pi' = \langle e_2, ..., e_k \rangle$. From Equation 5.7, we get

$$\begin{aligned} L(\pi) &= \max_{2 \le u \le k} \{ r(e_1), r(e_u) + \sum_{v=1}^{u-1} r'(e_v) \} \\ &= \max_{2 \le u \le k} \{ r(e_1), r'(e_1) + r(e_u) + \sum_{v=2}^{u-1} r'(e_v) \} \\ &= \max \{ r(e_1), r'(e_1) + \max_{2 \le u \le k} \{ r(e_u) + \sum_{v=2}^{u-1} r'(e_v) \} \} \\ &= \max \{ r(e_1), r'(e_1) + L(\pi') \}. \end{aligned} \tag{5.8}$$

Let $e_1 = \langle a_j, b_{j'} \rangle$ for $1 \leq j < j' \leq n$. It follows from Equation 5.8 that the length of a shortest path from node a_j to a_{n+1} or b_{n+1} with the first arc e_1 is $\max\{r(e_1), r'(e_1) + L^*_{b_{j'}}\}$, where $L^*_{b_{j'}}$ is the shortest path from node $b_{j'}$ to node a_{n+1} or b_{n+1}. Thus, a shortest path from node a_j to a_{n+1} or b_{n+1} is

$$\min_{1 \leq j < j' \leq n} \{\max\{r(a_j, b_{j'}), r'(a_j, b_{j'}) + L^*_{b_{j'}}\}, r(a_j, b_{n+1})\}.$$

Similarly, a shortest path from node b_j to a_{n+1} or b_{n+1} is

$$\min_{1 \leq j < j' \leq n} \{\max\{r(b_j, a_{j'}), r'(b_j, a_{j'}) + L^*_{a_{j'}}\}, r(b_j, a_{n+1})\},$$

where $L^*_{a_{j'}}$ is the shortest path from node $a_{j'}$ to a_{n+1} or b_{n+1}.

Thus, we get a dynamic programming recursion which is implemented using Equations 5.1 to 5.6. We shall now interpret the variables $f^*_{jj'}$, $g^*_{jj'}$, L_{1j}, and L_{2j}. For $1 \leq j \leq j' \leq n$,

$f^*_{jj'}$ is the length of a shortest path from node a_j when arc $\langle a_j, b_{j'+1} \rangle$ is chosen;

$g^*_{jj'}$ is the length of a shortest path from node b_j when arc $\langle b_j, a_{j'+1} \rangle$ is chosen;

L_{1j} is the length of a shortest path from node a_j; and

L_{2j} is the length of a shortest path from node b_j.

The length of a shortest path from node 0 is $\max\{-\infty, s_1 + L_{11}\} = (s_1 + L_{11})$ when arc $\langle 0, a_1 \rangle$ is chosen, and is $\max\{-\infty, s_2 + L_{21}\} = (s_2 + L_{21})$ when arc $\langle 0, b_1 \rangle$ is chosen. Hence, the length of a shortest path from node 0 is $\min\{s_1 + L_{11}, s_2 + L_{21}\}$.

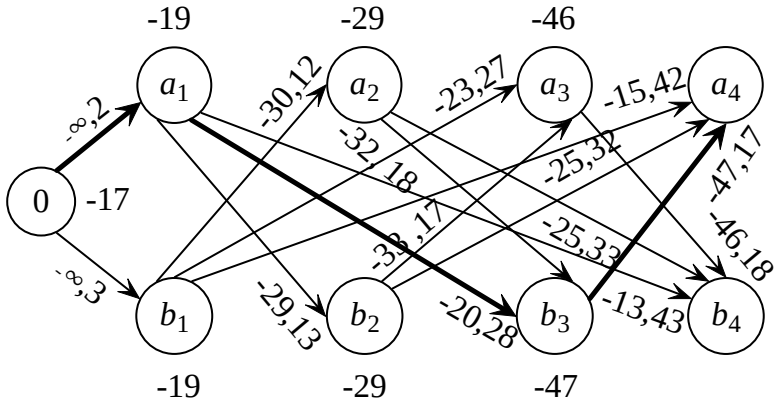

Figure 5.1: A network representation of the two-machine open shop problem

Example 5.1 *Consider the following problem involving* $n = 3$ *jobs with* $s_1 = 2$ *and* $s_2 = 3$.

j	1	2	3
t_{1j}	8	9	7
t_{2j}	2	6	8
d_j	42	50	64

Introduce a dummy job 4. The nodes are 0, a_1, ..., a_4 and b_1, ..., b_4. The corresponding graph and arc weights are shown in Figure 5.1. As we have discussed above the arc weights are computable functions of processing times, due dates and setup times. For example,

$$\begin{aligned} r(a_1, b_3) &= f_{1(3-1)} = f_{12} \\ &= \max\{t_{21} - d_1, t_{21} + t_{22} - d_2\} + s_2 + (t_{11} + t_{12}) \quad \text{from Equation 5.1} \\ &= \max\{2 - 42, 2 + 6 - 50\} + 3 + (8 + 9) = -20 \quad \text{and} \\ r'(a_1, b_3) &= \tau_{1(3-1)} + s_2 = \tau_{12} + s_2 \\ &= (8 + 9 + 2 + 6) + 3 = 28 \end{aligned}$$

The shortest paths are computed first from nodes a_3 and b_3, then from nodes a_2 and b_2 and so on. For example, the length of a shortest path from node a_1,

$$\begin{aligned} L^*_{a_1} &= \min\{\max\{r(a_1,b_2), r'(a_1,b_2) + L^*_{b_2}\}, \max\{r(a_1,b_3), r'(a_1,b_3) + L^*_{b_3}\}, r(a_1,b_4)\} \\ &= \min\{\max\{-29, 13 + (-29)\}, \max\{-20, 28 + (-47)\}, -13\} = -19 \end{aligned}$$

A shortest path from node 0 is $\langle\langle 0,a_1\rangle, \langle a_1,b_3\rangle, \langle b_3,a_4\rangle\rangle$. Arc $\langle 0,a_1\rangle$ represents setting up of machine 1, arc $\langle a_1,b_3\rangle$ represents batch $[1,2]$ and arc $\langle b_3,a_4\rangle$ represents batch $[3,3]$. Hence, an optimal solution is to start from machine 1 and use batching policy $(2,3)$. The corresponding sequence of operations is $(1,1)$, $(1,2)$, $(2,1)$, $(2,2)$, $(2,3)$, $(1,3)$.

We shall now discuss an $O(n)$ time implementation of the above dynamic programming recursion. Our implementation utilizes the results developed below.

5.1.2 Useful Facts

Proofs of the following results are given below.

Fact 5.1 *For $i \le j \le k$: (i) $f_{ik} \ge f_{ij} + \sum_{l=(j+1)}^{k} t_{1l}$; (ii) $f_{ik} \ge f_{jk} + \tau_{i(j-1)}$; (iii) $g_{ik} \ge g_{ij} + \sum_{l=(j+1)}^{k} t_{2l}$; and (iv) $g_{ik} \ge g_{jk} + \tau_{i(j-1)}$.*

Fact 5.2 *For $i \le n$: (i) $L_{1i} \ge \max_{i\le j\le n}\{\tau_{ij} - d_j\} + s_2$; and (ii) $L_{2i} \ge \max_{i\le j\le n}\{\tau_{ij} - d_j\} + s_1$.*

Fact 5.3 *For $i \le n$: (i) $L_{2i} + s_2 \ge L_{1i}$; and (ii) $L_{1i} + s_1 \ge L_{2i}$.*

Fact 5.4 *For $1 \le i < j \le n$: (i) $L_{1i} \ge L_{1j} + \tau_{i(j-1)}$; and (ii) $L_{2i} \ge L_{2j} + \tau_{i(j-1)}$.*

Proofs of Facts

Proof of Fact 1: Statement (i) follows from the relation $\max_{i\le u\le k}\{\sum_{l=i}^{u} t_{2l} - d_u\} + \sum_{l=i}^{k} t_{1l} \ge \max_{i\le u\le j}\{\sum_{l=i}^{u} t_{2l} - d_u\} + \sum_{l=i}^{j} t_{1l} + \sum_{l=j+1}^{k} t_{1l}$. Statement (ii) follows from the relation $\max_{i\le u\le k}\{\sum_{l=i}^{u} t_{2l} - d_u\} \ge \max_{j\le u\le k}\{\sum_{l=j}^{u} t_{2l} - d_u\} + \sum_{l=i}^{j-1} t_{2l}$. Statements (iii) and (iv) follow similarly. ■

Proof of Fact 2: Statements (i) and (ii) are clearly true for $i = n$. Suppose they are true for all $i \ge p$, for some p with $2 \le p < n$. Now let $i = p - 1$, and note that $L_{1i} = f^*_{ik}$ for some k, $i \le k \le n$. Statement (i) is proved for $i = p - 1$ by showing (a) $f^*_{ik} \ge \max_{i\le j\le k}\{\tau_{ij} - d_j\} + s_2$, and (b) $f^*_{ik} \ge \max_{k+1\le j\le n}\{\tau_{ij} - d_j\} + s_2$ if $k < n$.

To show (a), note that $\max_{i\le j\le k}\{\tau_{ij} - d_j\} = \tau_{ij'} - d_{j'}$ for some $i \le j' \le k$. Thus, $\max_{i\le j\le k}\{\tau_{ij} - d_j\} + s_2 = \sum_{l=i}^{j'} t_{2l} - d_{j'} + s_2 + \sum_{l=i}^{j'} t_{1l} \le \sum_{l=i}^{j'} t_{2l} - d_{j'} + s_2 + \sum_{l=i}^{k} t_{1l}$ $\le \max_{i\le j\le k}\{\sum_{l=i}^{j} t_{2l} - d_j\} + s_2 + \sum_{l-i}^{k} t_{1l} = f_{ik} \le f^*_{ik}$, as required.

To show (b), observe from (5.1) that for $k < n$, $f^*_{ik} \ge L_{2(k+1)} + s_2 + \tau_{ik}$. The induction hypothesis gives $L_{2(k+1)} \ge \max_{k+1\le j\le n}\{\tau_{(k+1)j} - d_j\} + s_1$, so $f^*_{ik} \ge \max_{k+1\le j\le n}\{\tau_{(k+1)j} - d_j\} + s_1 + s_2 + \tau_{ik} = \max_{k+1\le j\le n}\{\tau_{ij} - d_j\} + s_2 + s_1 \ge \max_{k+1\le j\le n}\{\tau_{ij} - d_j\} + s_2$, as required. Statement (ii) is proved similarly. ■

Proof of Fact 3: Statements (i) and (ii) are true for $i = n$. Suppose they are true for all $i \ge p$, for some p with $2 \le p \le n$. Let $i = p - 1$ and note that $L_{2i} = g^*_{ik}$ for some k, $i \le k \le n$. There are two cases: (a) $k = i$, and (b) $k > i$. Note that $g_{ii} + s_2 \ge f_{ii}$ and, by the induction hypothesis, $L_{1(i+1)} + s_1 \ge L_{2(i+1)}$.

In case (a), $L_{2i} + s_2 = g^*_{ii} + s_2 = \max\{g_{ii} + s_2, L_{1(i+1)} + \tau_{ii} + s_1 + s_2\} \geq \max\{f_{ii}, L_{2(i+1)} + \tau_{ii} + s_2\} = f^*_{ii} \geq L_{1i}$, as required.

In case (b), use Facts 5.1(iii) and 5.1(iv) to get $g_{ik} + s_2 \geq g_{ii} + s_2 \geq f_{ii}$ and $g_{ik} + s_2 \geq g_{(i+1)k} + \tau_{ii} + s_2$. Thus, $L_{2i} + s_2 = \max\{g_{ik} + s_2, L_{1(k+1)} + \tau_{ik} + s_1 + s_2\} \geq \max\{f_{ii}, g_{(i+1)k} + \tau_{ii} + s_2, L_{1(k+1)} + \tau_{ik} + s_1 + s_2\} = \max\{f_{ii}, \max\{g_{(i+1)k}, L_{1(k+1)} + \tau_{(i+1)k} + s_1\} + \tau_{ii} + s_2\} = \max\{f_{ii}, g^*_{(i+1)k} + \tau_{ii} + s_2\} \geq \max\{f_{ii}, L_{2(i+1)} + \tau_{ii} + s_2\} = f^*_{ii} \geq L_{1i}$, as required.

Statement (ii) is proved similarly. ■

Proof of Fact 4: Statements (i) and (ii) are true for $i = n - 1$. Suppose they are true for all $i \geq p$, for some p with $2 \leq p \leq n-1$. Let $i = p-1$ and note that $L_{1i} = f^*_{ik}$ for some k, $i \leq k \leq n$. There are three cases: (a) $k = n$, (b) $j \leq k < n$, and (c) $k < j$.

In case (a), $L_{1i} = f^*_{in} = f_{in} \geq f_{jn} + \tau_{i(j-1)}$, from Fact 5.1(ii). Thus $L_{1i} \geq f^*_{jn} + \tau_{i(j-1)} \geq L_{1j} + \tau_{i(j-1)}$.

In case (b), Fact 5.1(ii) gives $f_{ik} \geq f_{jk} + \tau_{i(j-1)}$, so $L_{1i} = f^*_{ik} = \max\{f_{ik}, L_{2(k+1)} + \tau_{ik} + s_2\} \geq \max\{f_{jk} + \tau_{i(j-1)}, L_{2(k+1)} + \tau_{jk} + \tau_{i(j-1)} + s_2\} = \max\{f_{jk}, L_{2(k+1)} + \tau_{jk} + s_2\} + \tau_{i(j-1)} = f^*_{jk} + \tau_{i(j-1)} \geq L_{1j} + \tau_{i(j-1)}$.

In case (c), $L_{1i} = f^*_{ik} \geq L_{2(k+1)} + \tau_{ik} + s_2$. If $k + 1 = j$, $L_{1i} \geq L_{2j} + \tau_{i(j-1)} + s_2 \geq L_{1j} + \tau_{i(j-1)}$, from Fact 5.3(i). If $k+1 < j$, the induction hypothesis gives $L_{2(k+1)} \geq L_{2j} + \tau_{(k+1)(j-1)}$, so $L_{1i} \geq L_{2j} + \tau_{(k+1)(j-1)} + \tau_{ik} + s_2 = L_{2j} + \tau_{i(j-1)} + s_2 \geq L_{1j} + \tau_{i(j-1)}$,

from Fact 5.3(i).

In all cases, statement (i) holds for $i = p - 1$.

The proof of statement(ii) is similar. ■

5.1.3 Useful Lemmas

We use the following concept.

Definition 5.1 *(i) Job k^* is* M1-critical *in batch $[i,j]$ if $k^* = \arg\max_{i \le k \le j}\{\sum_{l=i}^{k} t_{1l} - d_k\}$; (ii) job k^* is* M2-critical *in batch $[i,j]$ if $k^* = \arg\max_{i \le k \le j}\{\sum_{l=i}^{k} t_{2l} - d_k\}$.*

The following results permit restriction of the search space in determining an optimal policy. Lemma 5.1 implies that for jobs $i, \ldots, n$ there is an optimal policy whose initial batch extends at least as far as a critical job. Lemma 5.2 implies that an optimal initial batch for jobs $i, \ldots, n$ ends on or before the end of an optimal initial batch for jobs $i+1, \ldots, n$. Finally, Lemma 5.3 implies that if the initial batch $[i,j]$ is better than the initial batch $[i,j-1]$, then an optimal initial batch for jobs $i, \ldots, n$ extends at least to job j. Proofs of the Lemmas are deferred to the next subsection.

Lemma 5.1 *For $i \le j \le n$: (i) if job k^* is M1-critical in batch $[i,j]$, then $L_{2i} = \min_{k^* \le l \le n}\{g^*_{il}\}$; and (ii) if job k^* is M2-critical in batch $[i,j]$, then $L_{1i} = \min_{k^* \le l \le n}\{f^*_{il}\}$.*

Lemma 5.2 *For $1 \le i < j < l \le n$: (i) if $f^*_{(i+1)j} \le f^*_{(i+1)l}$, then $f^*_{ij} \le f^*_{il}$; and (ii) if $g^*_{(i+1)j} \le g^*_{(i+1)l}$, then $g^*_{ij} \le g^*_{il}$.*

In terms of the network representation, Lemma 5.2 provides a rule for *node elimination.* Consider $1 \leq i < j < l \leq n$. Suppose that the shortest paths from nodes a_{i+1}, ..., a_{n+1} and b_{i+1}, ..., b_{n+1} are computed and it is found that arc $\langle a_{i+1}, b_{l+1} \rangle$ is not a better choice than arc $\langle a_{i+1}, b_{j+1} \rangle$, i.e., the length of a shortest path containing arc $\langle a_{i+1}, b_{l+1} \rangle$ is not less than that of a shortest path containing arc $\langle a_{i+1}, b_{j+1} \rangle$. Statement (i) implies that arc $\langle a_i, b_{l+1} \rangle$ is not a better choice than arc $\langle a_i, b_{j+1} \rangle$. Applying the same statement again, since arc $\langle a_i, b_{l+1} \rangle$ is not a better choice than arc $\langle a_i, b_{j+1} \rangle$, arc $\langle a_{i-1}, b_{l+1} \rangle$ is not a better choice than arc $\langle a_{i-1}, b_{j+1} \rangle$. Proceeding similarly we can conclude that node b_{l+1} can be eliminated from further consideration.

Statement (ii) similarly suggests that if for $1 \leq i < j < l \leq n$, arc $\langle b_{i+1}, a_{l+1} \rangle$ is not a better choice than arc $\langle b_{i+1}, a_{j+1} \rangle$, then node a_{l+1} can be eliminated from further consideration.

The node elimination rule can thus be stated as follow: *if the longer arc is not better, then eliminate node.* More precisely, if for $1 \leq i < j < l \leq n$ a shortest path from node a_{i+1} with the first arc $\langle a_{i+1}, b_{l+1} \rangle$ is not shorter than a shortest path from node a_{i+1} with the first arc $\langle a_{i+1}, b_{j+1} \rangle$, then eliminate node b_{l+1}. Similarly, if for $1 \leq i < j < l \leq n$ a shortest path from node b_{i+1} with the first arc $\langle b_{i+1}, a_{l+1} \rangle$ is not shorter than a shortest path from node b_{i+1} with the first arc $\langle b_{i+1}, a_{j+1} \rangle$, then eliminate node a_{l+1}.

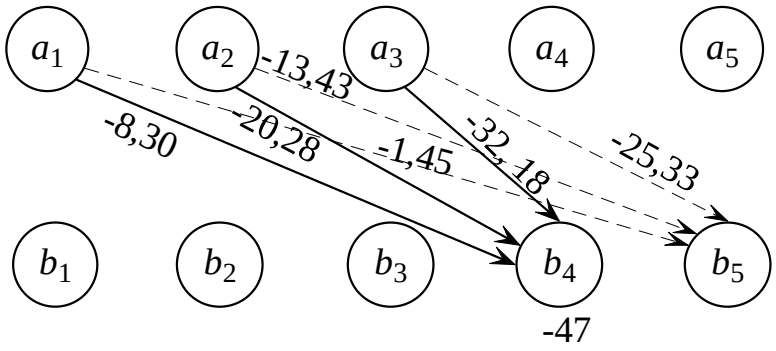

Figure 5.2: Elimination of node b_5

Example 5.2 *Consider the following problem involving* $n = 4$ *jobs with* $s_1 = 2$ *and* $s_2 = 3$.

j	1	2	3	4
t_{1j}	1	8	9	7
t_{2j}	1	2	6	8
d_j	30	42	50	64

The node elimination rule is illustrated in Figure 5.2 with $i = 2$, $j = 3$ and $l = 4$. When we compute a shortest path from node a_3, we find that the length of a shortest path is -25, if arc $\langle a_3, b_5 \rangle$ is chosen. However, if arc $\langle a_3, b_4 \rangle$ is chosen a shorter path is obtained with the length $= \max\{-32, 18 + (-47)\} = -29$. Hence, the longer arc $\langle a_3, b_5 \rangle$ is not better than the shorter arc $\langle a_3, b_4 \rangle$. Statement (i) of Lemma 5.2 implies that node b_5 can be eliminated from the computation of a shortest path from nodes a_2, a_1, etc. Observe that from node a_2 the length of a shortest path is -13, if arc $\langle a_2, b_5 \rangle$ is chosen and $\max\{-20, 28 + (-47)\} = -19$, if arc $\langle a_2, b_4 \rangle$ is chosen. From node a_1 the length of a shortest path is -1, if arc $\langle a_1, b_5 \rangle$ is chosen and $\max\{-8, 30 + (-47)\} = -8$, if arc $\langle a_1, b_4 \rangle$ is chosen. Hence, arcs $\langle a_2, b_4 \rangle$ and $\langle a_1, b_4 \rangle$

are better than arcs $\langle a_2, b_5 \rangle$ and $\langle a_1, b_5 \rangle$, respectively. Thus, elimination of node b_5 is justified.

Lemma 5.3 *For* $1 \leq i \leq l < j-1 \leq n-1$*: (i) if* $f^*_{i(j-1)} > f^*_{ij}$*, then* $f^*_{il} > f^*_{ij}$ *; and (ii) if* $g^*_{i(j-1)} > g^*_{ij}$*, then* $g^*_{il} > g^*_{ij}$.

In terms of the network representation, Lemma 5.3 provides a rule for *arc elimination.* Consider $1 \leq i \leq l < j-1 \leq n-1$, and consider the problem of computing a shortest path from node a_i. Suppose that arc $\langle a_i, b_j \rangle$ is a strictly worse choice than arc $\langle a_i, b_{j+1} \rangle$. Statement (i) implies that arc $\langle a_i, b_{l+1} \rangle$ is a strictly worse choice than arc $\langle a_i, b_{j+1} \rangle$. This means that every arc $\langle a_i, b_{j'} \rangle$ with $(i+1) \leq j' < j$ can be eliminated.

Statement (ii) similarly suggests that if for $1 \leq i \leq l < j-1 \leq n-1$, the arc $\langle b_i, a_j \rangle$ is a strictly worse choice than arc $\langle b_i, a_{j+1} \rangle$, then every arc $\langle b_i, a_{j'} \rangle$ with $(i+1) \leq j' < j$ can be eliminated.

The arc elimination rule can thus be stated as follow: *if the longer arc is better, then eliminate all shorter arcs.* More precisely, if for $1 \leq i \leq (n-2)$, $3 \leq j \leq n$, a shortest path from node a_i with the first arc $\langle a_i, b_{j+1} \rangle$ is shorter than a shortest path from node a_i with the first arc $\langle a_i, b_j \rangle$, then eliminate arc $\langle a_i, b_{j'} \rangle$ with $(i+1) \leq j' < j$. Similarly, if for $1 \leq i \leq (n-2)$, $3 \leq j \leq n$, a shortest path from node b_i with the first arc $\langle b_i, a_{j+1} \rangle$ is shorter than a shortest path from node b_i with the first arc $\langle b_i, a_j \rangle$, then eliminate all arcs $\langle b_i, a_{j'} \rangle$ with $(i+1) \leq j' < j$.

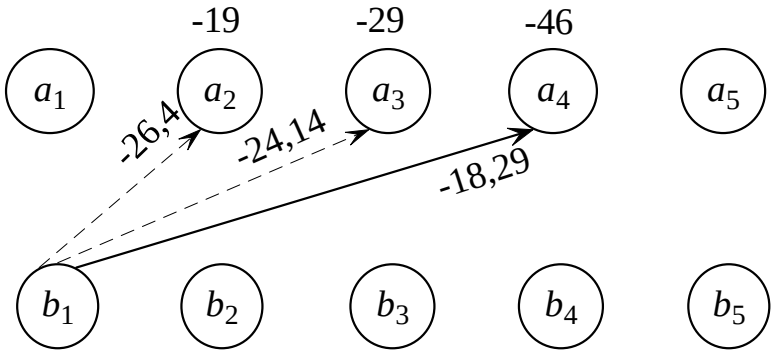

Figure 5.3: Elimination of arc $\langle b_1, a_2 \rangle$

Considering the data shown in Example 5.2, we illustrate the arc elimination rule in Figure 5.3 with $i = 1$, $j = 3$ and $l = 1$. When we compute a shortest path from node b_1, we find that the length of a shortest path is $\max\{-18, 29 + (-46)\} = -17$, if arc $\langle b_1, a_4 \rangle$ is chosen and $\max\{-24, 14 + (-29)\} = -15$, if arc $\langle b_1, a_3 \rangle$ is chosen. Hence, the longer arc $\langle b_1, a_4 \rangle$ is better that the shorter arc $\langle b_1, a_3 \rangle$. Statement (ii) of Lemma 5.3 implies that arc $\langle b_1, a_2 \rangle$ can be eliminated. Since, the length of a shortest path is $\max\{-26, 4 + (-19)\} = -15$, if arc $\langle b_1, a_2 \rangle$ is chosen, the elimination of arc $\langle b_1, a_2 \rangle$ is justified.

Proofs of the Lemmas

Proof of Lemma 1: (i) Suppose the result does not hold. Then, there is a batch $[i, j]$ and an M1-critical job k^* in $[i, j]$ such that $L_{2i} < \min_{k^* \le l \le n}\{g^*_{il}\}$. This implies the existence of k, $i \le k \le k^* - 1$, such that $L_{2i} = g^*_{ik} < \min_{k^* \le l \le n}\{g^*_{il}\}$, so $g^*_{ik} < g^*_{ik^*}$.

If $k^* = n$ (requiring $j = n$), $g^*_{ik} = \max\{g_{ik}, L_{1(k+1)} + \tau_{ik} + s_1\} < g_{in}$. This implies

$L_{1(k+1)} + \tau_{ik} + s_1 < g_{in} = \max_{i \le j \le n}\{\sum_{l=i}^{j} t_{1l} - d_j\} + s_1 + \sum_{l=i}^{n} t_{2l} = \sum_{l=i}^{n} t_{1l} - d_n + s_1 + \sum_{l=i}^{n} t_{2l}$ (since job n is M1-critical). Thus, $L_{1(k+1)} < \tau_{(k+1)n} - d_n$, which contradicts Fact 5.2(i).

If $k^* < n$, $g^*_{ik} < g^*_{ik^*}$ gives $\max\{g_{ik}, L_{1(k+1)} + \tau_{ik} + s_1\} < \max\{g_{ik^*}, L_{1(k^*+1)} + \tau_{ik^*} + s_1\}$, so $L_{1(k+1)} + \tau_{ik} + s_1 < \max\{g_{ik^*}, L_{1(k^*+1)} + \tau_{ik^*} + s_1\}$. Now $g_{ik^*} = \max_{i \le j \le k^*}\{\sum_{l=i}^{j} t_{1l} - d_j\} + s_1 + \sum_{l=i}^{k^*} t_{2l} = \sum_{l=i}^{k^*} t_{1l} - d_{k^*} + s_1 + \sum_{l=i}^{k^*} t_{2l}$, because k^* is M1-critical. If $L_{1(k+1)} + \tau_{ik} + s_1 < g_{ik^*}$ we have $L_{1(k+1)} < \tau_{(k+1)k^*} - d_{k^*}$, which contradicts Fact 5.2(i). If $L_{1(k+1)} + \tau_{ik} + s_1 < L_{1(k^*+1)} + \tau_{ik^*} + s_1$ we have $L_{1(k+1)} < L_{1(k^*+1)} + \tau_{(k+1)k^*}$, which contradicts Fact 5.4(i).

These contradictions prove (i). Statement (ii) is proved similarly. ■

Proof of Lemma 2: (i) Since $f^*_{(i+1)l} = \max\{f_{(i+1)l}, L_{2(l+1)} + \tau_{(i+1)l} + s_2\} \ge f^*_{(i+1)j}$, either (a) $f_{(i+1)l} \ge f^*_{(i+1)j}$, or (b) $L_{2(l+1)} + \tau_{(i+1)l} + s_2 \ge f^*_{(i+1)j}$.

In case (a) $f_{(i+1)l} \ge f^*_{(i+1)j} \ge L_{2(j+1)} + \tau_{(i+1)j} + s_2$. From Fact 5.1(i) and (ii) $f_{il} \ge \max\{f_{ij}, f_{(i+1)l}\} \ge f^*_{ij}$. Hence, $f^*_{il} \ge f_{il} \ge f^*_{ij}$, as required.

In case (b), $L_{2(l+1)} + \tau_{(i+1)l} + s_2 \ge f^*_{(i+1)j} \ge L_{2(j+1)} + \tau_{(i+1)j} + s_2$ so $L_{2(l+1)} + \tau_{il} + s_2 \ge L_{2(j+1)} + \tau_{ij} + s_2$. Fact 5.1(i) gives $f_{il} \ge f_{ij}$, so $f^*_{il} = \max\{f_{il}, L_{2(l+1)} + \tau_{il} + s_2\} \ge \max\{f_{ij}, L_{2(j+1)} + \tau_{ij} + s_2\} = f^*_{ij}$, as required.

Statement (ii) is proved similarly. ■

Proof of Lemma 3: (i) Since $f^*_{i(j-1)} = \max\{f_{i(j-1)}, L_{2j} + \tau_{i(j-1)} + s_2\} > f^*_{ij}$, either (a) $f_{i(j-1)} > f^*_{ij}$, or (b) $L_{2j} + \tau_{i(j-1)} + s_2 > f^*_{ij}$. However, Fact 5.1(i) gives $f_{i(j-1)} \le$

$f_{ij} \leq f^*_{ij}$, so case (a) cannot occur. We thus have case (b). Now, Fact 5.4(ii), gives $L_{2(l+1)} \geq L_{2j} + \tau_{(l+1)(j-1)}$, so $L_{2(l+1)} + \tau_{il} + s_2 \geq L_{2j} + \tau_{i(j-1)} + s_2 > f^*_{ij}$. Therefore, $f^*_{il} = \max\{f_{il}, L_{2(l+1)} + \tau_{il} + s_2\} > f^*_{ij}$. This proves (i).

Statement (ii) follows similarly. ■

5.1.4 The Algorithm

Algorithm 1 below is a streamlined dynamic programming recursion scheme. Each iteration considers the problem for jobs $i, \ldots, n$, where i is initialized by $n-1$ and is reduced by 1 in each iteration. The iteration determines implicitly two schedules, σ_{1i} and σ_{2i}: schedule σ_{1i} is optimal, starting from machine 1, while schedule σ_{2i} is optimal starting from machine 2. One of these two schedules is optimal for the problem with jobs $i, \ldots, n$. The first batch of σ_{1i} is $[i, j_1]$. This information is stored in array $Succ$ by setting $Succ(1, i) = j_1$. For $j_1 < n$, the other batches of σ_{1i} are same as those given by $\sigma_{2(j_1+1)}$. Similarly, $[i, j_2]$ is the first batch of σ_{2i}, this information is stored by setting $Succ(2, i) = j_2$. For $j_2 < n$, other batches of σ_{2i} are same as those given by $\sigma_{1(j_2+1)}$.

At the beginning of the iteration, k^*_2 is M2-critical in $[i+1, j_1]$, k^*_1 is M1-critical in batch $[i+1, j_2]$, $e_1 = \sum_{l=(i+1)}^{k^*_1} t_{1l} - d_{k^*_1}$, and $e_2 = \sum_{l=(i+1)}^{k^*_2} t_{2l} - d_{k^*_2}$.

Since k^*_2 is M2-critical for batch $[i+1, j_1]$, at least one of i and k^*_2 is M2-critical for batch $[i, j_1]$. If i is critical, $t_{2i} - d_i \geq e_2 + t_{2i}$ and $f_{ij_1} = f_{(i+1)j_1} + t_{1i} + t_{2i} - d_i - e_2$, while if k^*_2 is critical, $t_{2i} - d_i \leq e_2 + t_{2i}$ and $f_{ij_1} = f_{(i+1)j_1} + t_{1i} + t_{2i}$. Hence, f_{ij_1} can be

computed from $f_{(i+1)j_1}$ in constant time. Similarly, g_{ij_2} can be computed from $g_{i(j_2+1)}$ in constant time. These computations are implemented in Procedure 1.

At the beginning of the iteration, $[i+1, j_1]$ is the optimal first batch of schedule $\sigma_{1(i+1)}$ and $[i+1, j_2]$ is the optimal first batch of schedule $\sigma_{2(i+1)}$. Procedure 2 searches for j_1^* and j_2^* so that for $m = 1, 2$, batch $[i, j_m^*]$ qualifies for being an optimal first batch of schedule σ_{mi}. From Lemma 5.1(ii), $L_{1i} = \min_{k_2^* \le l \le n}\{f_{il}^*\}$, so $j_1^* \ge k_2^*$. Since $f_{(i+1)j_1}^* \le f_{(i+1)l}^*$ for all $l \ge j_1$, it follows from Lemma 5.2(i) that $f_{ij_1}^* \le f_{il}^*$ for all $l \ge j_1$, hence $j_1^* \le j_1$. From Lemma 5.3(i) it follows that if $f_{i(j-1)}^* > f_{ij}^*$ for some j, then $f_{il}^* > f_{ij}^*$ for all $l < j$, so $j_1^* \ge j$.

In summary: $j_1^* = \min\{j : k_2^* \le j \le j_1 \text{ and } f_{ij}^* \le f_{i(j+1)}^* \le \ldots \le f_{ij_1}^*\}$. Similarly, $j_2^* = \min\{j : k_1^* \le j \le j_2 \text{ and } g_{ij}^* \le g_{i(j+1)}^* \le \ldots \le g_{ij_2}^*\}$. Hence, each of j_1^* and j_2^* can be obtained by a sequential search process. Such processes are implemented in Procedure 2.

Finally, Procedure 3 identifies an optimal schedule. If $L_{11} + s_1 \le L_{21} + s_2$, the process starts on machine 1; otherwise the process starts on machine 2. The batches are constructed using the information stored in array *Succ*.

Input: $t_{1i}, t_{2i}, d_i \ \forall\ 1 \leq i \leq n$, s_1 and s_2.
Output: An optimal schedule.
$j_1 = j_2 = k_1^* = k_2^* = n$
$e_1 = t_{1n} - d_n : e_2 = t_{2n} - d_n$
$Succ(1, n) = Succ(2, n) = n$
Compute f_{nn}, g_{nn}, τ_{nn}, f_{nn}^*, g_{nn}^*, L_{1n}, L_{2n}
For $i = (n-1)$ down to 1 do
    Update critical jobs and f_{ij_1}, g_{ij_2}, e_1, and e_2 using Procedure 1
    $\tau_{ij_1} = \tau_{(i+1)j_1} + t_{1i} + t_{2i} : \tau_{ij_2} = \tau_{(i+1)j_2} + t_{1i} + t_{2i}$
    Update j_1, j_2, $f_{ij_1}^*$ and $g_{ij_2}^*$ from Procedure 2
    $Succ(1, i) = j_1 : Succ(2, i) = j_2$
    $L_{1i} = f_{ij_1}^* : L_{2i} = f_{ij_2}^*$
EndFor
Construct an optimal schedule using Procedure 3

Algorithm 1: Open Shop $L_{\max}$ Problem

If $t_{2i} - d_i \geq e_2 + t_{2i}$ then
    $k_2^* = i : f_{ij_1} = f_{(i+1)j_1} + t_{1i} + t_{2i} - d_i - e_2 : e_2 = t_{2i} - d_i$
Else
    $f_{ij_1} = f_{(i+1)j_1} + t_{1i} + t_{2i} : e_2 = e_2 + t_{2i}$
EndIf
If $t_{1i} - d_i \geq e_1 + t_{1i}$ then
    $k_1^* = i : g_{ij_2} = g_{(i+1)j_2} + t_{1i} + t_{2i} - d_i - e_1 : e_1 = t_{1i} - d_i$
Else
    $g_{ij_2} = g_{(i+1)j_2} + t_{1i} + t_{2i} : e_1 = e_1 + t_{1i}$
EndIf

Procedure 1: Update critical jobs and f_{ij_1}, g_{ij_2}, e_1 and e_2.

Compute $f^*_{ij_1}$, $g^*_{ij_2}$
$j^*_1 = j_1 : j^*_2 = j_2$
Do while $j^*_1 > k^*_2$
    $f_{i(j^*_1-1)} = f_{ij^*_1} - t_{1j^*_1} : \tau_{i(j^*_1-1)} = \tau_{ij^*_1} - t_{1j^*_1} - t_{2j^*_1}$
    Compute $f^*_{i(j^*_1-1)}$
    If $f^*_{i(j^*_1-1)} \leq f^*_{ij^*_1}$ then $j^*_1 = j^*_1 - 1$ else Exit Do
Loop
Do while $j^*_2 > k^*_1$
    $g_{i(j^*_2-1)} = g_{ij^*_2} - t_{2j^*_2} : \tau_{i(j^*_2-1)} = \tau_{ij^*_2} - t_{1j^*_2} - t_{2j^*_2}$
    Compute $g^*_{i(j^*_2-1)}$
    If $g^*_{i(j^*_2-1)} \leq g^*_{ij^*_2}$ then $j^*_2 = j^*_2 - 1$ else Exit Do
Loop
$j_1 = j^*_1 : j_2 = j^*_2$

Procedure 2: Update j_1, j_2, $f^*_{ij_1}$ and $g^*_{ij_2}$

If $L_{11} + s_1 \leq L_{21} + s_2$ then $cm = 1$ else $cm = 2$.
The process starts from machine cm.
$i = 1$
Do
    $j = Succ(cm, i)$
    $[i, j]$ is a batch
    $i = j + 1$
    If $cm = 1$ then $cm = 2$ else $cm = 1$
Loop until $i > n$

Procedure 3: Construct an optimal schedule.

5.1.5 Complexity of the Algorithm

Procedure 1 has constant complexity and is called $(n-1)$ times. Throughout Algorithm 1 the values of j^*_1, j^*_2, k^*_1 and k^*_2 decrease monotonically, so each of the conditions $j^*_1 > k^*_2$ and $j^*_2 > k^*_1$ is satisfied at most $(n-1)$ times. Therefore, while the number of computations performed in successive calls to Procedure 2 may vary with

the main iteration count, the total number of computations performed in all calls to Procedure 2 is $O(n)$. Finally, Procedure 3 has time complexity $O(n)$. This proves:

Theorem 5.1 *Algorithm 1 solves the one-operator two-machine open shop problem with maximum lateness objective in $O(n)$ computations after due date sorting.*

In terms of the network representation, the running time of the algorithm reduces from $O(n^2)$ to $O(n)$ due to the following reasons: (i) the arc weights are generated on an "as needed" basis; (ii) every arc weight is generated from a previously generated arc weight in constant time; and (iii) each comparison between two arcs results in a node elimination or an arc elimination. If the longer arc is not better, we eliminate a node and if the longer arc is better we eliminate some arcs.

We shall now illustrate Algorithm 1 with an example.

5.1.6 An Example

Example 5.3 *Consider the following problem involving $n = 5$ jobs with $s_1 = 2$ and $s_2 = 3$.*

j	1	2	3	4	5
t_{1j}	1	1	8	9	7
t_{2j}	6	1	2	6	8
d_j	25	30	42	50	64

In the following we show all the $f^*_{ij_1}$ and $g^*_{ij_2}$ computed. Observe that $g^*_{3,3}$ is not computed because since job 4 is M1-critical in batch [3,4], we get from Lemma 5.1 that $L_{2,3} = \min\{g^*_{3,4}, g^*_{3,5}\}$. Quantity $f^*_{3,5}$ is not computed because from Lemma 5.2,

$f^*_{4,4} \leq f^*_{4,5}$ implies $f^*_{3,4} \leq f^*_{3,5}$ (node elimination). For similar reasons none of $f^*_{1,3}$, $f^*_{1,4}$, $f^*_{1,5}$, $f^*_{2,5}$, $g^*_{1,5}$, $g^*_{2,5}$ and $g^*_{3,5}$ are computed. Finally, $g^*_{2,2}$ is not computed because from Lemma 5.3, $g^*_{2,3} > g^*_{2,4}$ implies $g^*_{2,2} > g^*_{2,4}$ (arc elimination).

i	k^*_2	j_1	$f^*_{ij_1}$	L_{1i}	k^*_1	j_2	$g^*_{ij_2}$	L_{2i}
5	5	5	-46	-46	5	5	-47	-47
4	4	5	-25		4	5	-25	
4		4	-29	-29		4	-29	-29
3	3	4	-19		4	4	-19	-19
3		3	-16	-19				
2	2	4	-8		2	4	-17	
		3	-14			3	-15	-17
		2	-14	-14				
1	1	2	-7		1	4	-7	
		1	-7	-7		3	-8	
						2	-8	
						1	-5	-8

Since $L_{11} + s_1 = -5 \leq -5 = L_{21} + s_2$, an optimal schedule with $L_{\max} = -5$ starts on machine 1. The schedule has batches [1,1], [2,4] and [5,5].

5.2 The Flow Shop Problem

Define the following for $1 \leq i \leq j \leq n$:

$$f_{ij} = \max_{i \leq k \leq j} \{ \sum_{l=i}^{k} t_{2l} - d_k \} + s_1 + s_2 + \sum_{l=i}^{j} t_{1l} \tag{5.9}$$

$$f^*_{ij} = \begin{cases} f_{in} & \text{for } j = n \\ \max\{f_{ij}, L_{j+1} + \tau_{ij} + s_1 + s_2\} & \text{for } j < n \end{cases} \tag{5.10}$$

$$L_i = \min_{i \leq j \leq n} \{f^*_{ij}\}. \tag{5.11}$$

These quantities have a simple interpretation: given that the first setup for job i is started at time zero on machine 1, f_{ij} is the maximum lateness of jobs in batch

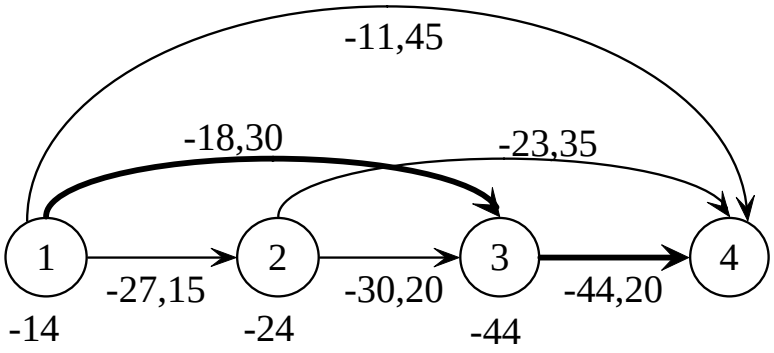

Figure 5.4: A network representation of the two machine flow shop problem

$[i,j]$, f^*_{ij} is the maximum lateness of jobs $i,\ldots,n$, given that $[i,j]$ is a batch, and L_i is the optimal maximum lateness of jobs $i,\ldots,n$. The minimum value of maximum lateness is $L^*_{\max} = L_1$.

A network representation for the flow shop case is much simpler than in the open shop case. Introduce a dummy job $(n+1)$. Define a directed network $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ is the node-set and $\mathbf{E}$ is the arc-set. For each operation j, there is a node j. For each pair of nodes j and j' with $1 \leq j < j' \leq (n+1)$, there is an arc $\langle j, j' \rangle$ which represents processing of batch $[j, j'-1]$ and is associated with weights $r(j,j') = f_{j(j'-1)}$ and $r'(j,j') = \tau_{j(j'-1)} + s_1 + s_2$. For example, suppose that there are three jobs 1, 2 and 3. Introduce a dummy job 4. The nodes are 1, 2, 3 and 4. The corresponding graph is shown in Figure 5.4.

The length of a path $\pi = \langle e_1, e_2, ..., e_k \rangle$ is $L(\pi) = \max_{1 \leq u \leq k} \{ r(e_u) + \sum_{v=1}^{u-1} r'(e_v) \}$ as defined in Equation 5.7. The length of π represents the maximum lateness when all the batches represented by π are carried out starting at time zero. The problem

is to find a shortest path from node 1 to $(n+1)$.

Since the length of the path is defined in exactly the same way as it is in the open shop case, we get $L(\pi) = \max\{r(e_1),\, r'(e_1) + L(\pi')\}$ as given in Equation 5.8. Thus, proceeding in the same way as we did in the open shop case, we get that for $1 \le j \le n$, the length of a shortest path from node j to $(n+1)$ is

$$\min_{1 \le j < j' \le n} \{\max\{r(j,j'), r'(j,j') + L^*_{j'}\}, r(j,n+1)\}.$$

where $L^*_{j'}$ is the shortest path from node j' to $(n+1)$.

Thus, we get a dynamic programming recursion which is implemented using Equations 5.9-5.11. In Equations 5.9-5.11, for $1 \le j \le j' \le n$, $f^*_{jj'}$ is the length of a shortest path from node j when arc $\langle j, j'+1\rangle$ is chosen; and L_j is the length of a shortest path from node j. Hence, an optimal solution is L_1.

Considering the data shown in Example 5.1, we illustrate the network representation in Figure 5.4. There are three jobs 1, 2 and 3. Introduce a dummy job 4. The nodes are 1, 2, 3, and 4. As in the open shop case, the arc weights are computable functions of processing times, due dates and setup times. For example,

$$\begin{aligned}
r(1,3) &= f_{1(3-1)} = f_{12} \\
&= \max\{t_{21} - d_1, t_{21} + t_{22} - d_2\} + s_1 + s_2 + (t_{11} + t_{12}) \quad \text{from Equation 5.9} \\
&= \max\{2 - 42, 2 + 6 - 50\} + 2 + 3 + (8+9) = -18 \quad \text{and} \\
r'(1,3) &= \tau_{1(3-1)} + s_1 + s_2 = \tau_{12} + s_1 + s_2 \\
&= (8 + 9 + 2 + 6) + 2 + 3 = 30
\end{aligned}$$

The shortest paths are computed first from nodes 3, then from node 2, and so on. For example, the length of a shortest path from node 1,

$$\begin{aligned} L_1^* &= \min\{\max\{r(1,2), r'(1,2) + L_2^*\}, \max\{r(1,3), r'(1,3) + L_3^*\}, r(1,4)\} \\ &= \min\{\max\{-27, 15 + (-24)\}, \max\{-18, 30 + (-44)\}, -11\} = -14 \end{aligned}$$

A shortest path from node 1 is $\langle\langle 1,3\rangle, \langle 3,4\rangle\rangle$. Arc $\langle 1,3\rangle$ represents batch $[1,2]$ and arc $\langle 3,4\rangle$ represents batch $[3,3]$. Hence, an optimal solution is to use batching policy $(2,3)$. The corresponding sequence of operations is $(1,1)$, $(1,2)$, $(2,1)$, $(2,2)$, $(1,3)$, $(2,3)$.

However, like the open shop case, the flow shop case is also solvable in $O(n)$ time. This is discussed below.

Definition 5.2 *A job k^* is a* critical job *in batch $[i,j]$ if $k^* = \arg\max_{i \le k \le j}\{\sum_{l=i}^{k} t_{2l} - d_k\}$.*

Lemmas 5.4, 5.5 and 5.6 below are similar to Lemmas 5.1, 5.2 and 5.3 respectively. Their proofs are also similar.

Lemma 5.4 *If job k^* is a critical job in batch $[i,j]$, then $L_i = \min_{k^* \le l \le n}\{f_{il}^*\}$.*

Lemma 5.5 *For $1 \le i < j < l \le n$: if $f_{(i+1)j}^* \le f_{(i+1)l}^*$, then $f_{ij}^* \le f_{il}^*$.*

In terms of the network representation, Lemma 5.5 provides a rule for *node elimination*. Consider $1 \le i < j < l \le n$. If arc $\langle i+1, l+1\rangle$ is a not better choice than arc $\langle i+1, j+1\rangle$, then node $(l+1)$ can be eliminated.

Lemma 5.6 *For $1 \le i \le l < j - 1 \le n - 1$: if $f_{i(j-1)}^* > f_{ij}^*$, then $f_{il}^* > f_{ij}^*$.*

In terms of the network representation, Lemma 5.6 provides a rule for *arc elimination.* Consider $1 \leq i \leq l < j-1 \leq n-1$. If arc $\langle i, j \rangle$ is a strictly worse choice than arc $\langle i, j+1 \rangle$, then every arc $\langle i, j' \rangle$ with $(i+1) \leq j' \leq j$ can be eliminated.

These lemmas lead to the following Algorithm 2 for the case of a flow shop. Justification is similar to that for Algorithm 1.

Theorem 5.2 *Algorithm 2 solves the one-operator two-machine flow shop problem with maximum lateness objective in $O(n)$ time after due date sorting.*

Input: $t_{1i}, t_{2i}, d_i \ \forall\ 1 \leq i \leq n$, s_1 and s_2.
Output: An optimal schedule.
$j = k^* = n : e = t_{2n} - d_n : Succ(n) = n$
Compute f_{nn}, τ_{nn}, f^*_{nn}, L_n
For $i = (n-1)$ down to 1 do
    If $t_{2i} - d_i \geq e + t_{2i}$ then
        $k^* = i : f_{ij} = f_{(i+1)j} + t_{1i} + t_{2i} - d_i - e : e = t_{2i} - d_i$
    Else
        $f_{ij} = f_{(i+1)j} + t_{1i} + t_{2i} : e = e + t_{2i}$
    EndIf
    $\tau_{ij} = \tau_{(i+1)j} + t_{1i} + t_{2i}$: Compute $f^*_{ij} : j^* = j$
    Do while $j^* > k^*$
        $f_{i(j^*-1)} = f_{ij^*} - t_{1j^*} : \tau_{i(j^*-1)} = \tau_{ij^*} - t_{1j^*} - t_{2j^*}$
        Compute $f^*_{i(j^*-1)}$
        If $f^*_{i(j^*-1)} \leq f^*_{ij^*}$ then $j^* = j^* - 1$ else Exit Do
    Loop
    $j = j^* : Succ(i) = j : L_i = f^*_{ij}$
EndFor
$i = 1$
Do
    $j = Succ(i)$: $[i, j]$ is a batch: $i = j + 1$
Loop until $i > n$.

Algorithm 2: Flow Shop $L_{\max}$ Problem

5.3 Summary

In this chapter we consider problems $1O2|s_i|L_{\max}$ and $1F2|s_i|L_{\max}$. Many existing dynamic programming recursions (Gerodimos et al. [45, 46], Ghosh and Gupta [48] and Webster and Baker [121]) solve the open shop case in $O(n^2)$ time. It is not known whether the flow shop case can be solved similarly. The dynamic programming scheme developed in Chapter 4 solves both the open shop and flow shop cases. However, the running time is again $O(n^2)$. In this chapter we develop a dynamic programming recursion which applies to both the open shop and flow shop cases and requires $O(n)$ time after due date sorting. Since the single machine problem $1|s_1, F = 1|\delta$ is a special case of the problem $1F2|s_i|\delta$, (see Section 3.5) we immediately get an improvement of the running time of the dynamic programming recursion for the problem $1|s_1, F = 1|L_{\max}$ given by Webster and Baker [121].

We give a network representation of the algorithm. We show that the algorithm resembles the shortest path algorithm if the lengths of arcs and paths are defined in a particular way. In terms of the network representation, the running time of the algorithm reduces from $O(n^2)$ to $O(n)$ due to the following reasons: (i) the arc weights are generated on an "as needed" basis; (ii) every arc weight is generated from a previously generated arc weight in constant time; and (iii) each comparison between two arcs results in a node elimination or an arc elimination. If the longer arc is not better, we eliminate a node and if the longer arc is better we eliminate some arcs.

Chapter 6

Two-Machine Problem with The Weighted Number of Tardy Jobs Objective

In this chapter we shall consider problems $1Om2|s_i|\sum w_jU_j$ and $1F2|s_i|\sum w_jU_j$. As we have discussed in Section 3.5, the problem $1|s_1,\ F=1|\sum w_jU_j$ is a special case of the problem $1F2|s_i|\sum w_jU_j$, and Hochbaum and Landy [58] show that the problem $1|s_1,\ F=1|\sum w_jU_j$ is $\mathcal{NP}$-hard. Hence, the problem $1F2|s_i|\sum w_jC_j$ is $\mathcal{NP}$-hard. However, when all $w_j=1$, the problem $1|s_1,\ F=1|\sum U_j$ is polynomially solvable [24, 58]. In contrast, our problems are $\mathcal{NP}$-hard even when all $w_j=1$. We shall show that even for two machines both the flow shop and open shop problems with the objective $\sum U_j$ are $\mathcal{NP}$-hard. Furthermore, we develop pseudo-polynomial dynamic programming algorithms for both the flow shop and open shop problems with objective $\sum w_jU_j$. As we have discussed in Section 1.5, a pseudo-polynomial algorithm is the one whose running time is polynomial under unary encoding of data

and exponential under binary encoding of data.

There are two styles of algorithms for the single machine scheduling problems with weighted number of tardy jobs objective. First, Lawler and Moore [77] present a pseudo-polynomial dynamic programming recursion which runs in time $O(n \sum t_j)$ where t_j is the processing time of job j. Sahni [104] presents another pseudo-polynomial dynamic programming recursion which runs in time $O(n \sum w_j)$.

Both of the above cited algorithms address the following question at the j-th stage: given jobs $1, ..., j$, does there exist a feasible schedule that has completion time of the last early job equal to t and the weighted number of tardy jobs equal to w? It is not necessary to enumerate over all t *and* all w. Instead, it is sufficient to enumerate over all t *or* all w. Lawler and Moore [77] enumerate over all t. If for some j and t, we have w^* equal to the least value of w such that the answer is "yes", then the information is stored by setting $h_j(t) = w^*$. Sahni [104] enumerates over all w. If for some j and w, we have t^* equal to the least value of t such that the answer is "yes", then the information is stored by setting $h_j(w) = t^*$. In dynamic programming terminology, Lawler and Moore [77] use the state variable t while Sahni [104] uses the state variable w. One advantage of using w as a state variable instead of t is that if all $w_j = 1$, the algorithm's running time is $O(n^2)$ because $\sum w_j = n$. Thus, the algorithm is polynomial if all $w_j = 1$. Sahni [104] shows that another advantage of using w as a state variable instead of t is that the algorithm can be used to get a

polynomial-time approximation scheme for the weighted case.

Hochbaum and Landy [58] extend the approach of Lawler and Moore [77] to the problem $1|s_1,\ F = 1|\sum w_j U_j$. Recall that the problem $1|s_1,\ F = 1|\sum w_j U_j$ involves processing jobs in batches. The completion time of a job is given by the completion time of the batch in which the job is processed. Hochbaum and Landy [58] incorporate a new state variable, d, that represents the earliest due date of the last early batch. Hochbaum and Landy [58] give an $O(n^2 \min\{d_{\max}, \sum t_j + ns_1\})$ time algorithm for the weighted case and another $O(n^4)$ time algorithm for the case with all $w_j = 1$. Brucker and Kovalyov [24] present another algorithm for the problem $1|s_1,\ F = 1|\sum w_j U_j$ which runs in time $O(n^2 \sum w_j)$.

Both the algorithms of Hochbaum and Landy [58] and Brucker and Kovalyov [24] address the following question at the j-th stage: given jobs $1, ..., j$, does there exist a feasible schedule that has the completion time of the last early job equal to t, the weighted number of tardy jobs equal to w and the earliest due date of the last early batch equal to d? Hochbaum and Landy [58] use state variables t and d, while Brucker and Kovalyov [24] use state variables w and d. One advantage of using w as a state variable instead of t is again that if all $w_j = 1$, the running time is $O(n^3)$. Hence, for the case of all $w_j = 1$, the algorithm runs in polynomial time. In this way, Brucker and Kovalyov [24] reduce the running time for the problem $1|s_1,\ F = 1|\sum U_j$ from $O(n^4)$ proposed by Hochbaum and Landy [58]. Brucker and Kovalyov [24] show that

another advantage of using w as a state variable instead of t is that the algorithm can be used to get a polynomial-time approximation scheme for the weighted case.

In this chapter we shall further extend each style of algorithms to both the open shop and flow shop cases. We replace the state variable d by a new state variable defined below. We report computational experience for problems having up to 100 jobs.

Gerodimos et al. [45, 46] have developed an $O(nd_{\max}^m)$ time algorithm for the single machine, m-operation problem a restricted case of which is equivalent to the one-operator open shop problem.

6.1 Proof of $\mathcal{NP}$-Hardness

Theorem 6.1 *Both* $1O2|s_i|\sum U_j$ *and* $1F2|s_i|\sum U_j$ *are* $\mathcal{NP}$*-hard.*

Proof: Given an instance of the partition problem with set of integers $\mathbf{A} = \{a_1, ..., a_k\}$, $\sum_{a_l \in \mathbf{A}} a_l = 2b$, we define an instance each of $1O2|s_i|\sum U_j$ and $1F2|s_i|\sum U_j$ as follows:

$$
\begin{array}{lcll}
n & = & 4k & \\
s_1 = s_2 & = & 2kb & \\
t_{1j} & = & \begin{cases} 2b - a_j & \text{for } j = 1, 2, ...k \\ 2b & \text{for } j = (k+1), ..., 2k \\ 0 & \text{for } j = (2k+1), ..., 4k \end{cases} & \\
t_{2j} & = & \begin{cases} 2b + 2a_j & \text{for } j = 1, 2, ...k \\ 2b & \text{for } j = (k+1), ..., 2k \\ 0 & \text{for } j = (2k+1), ..., 4k \end{cases} & \\
d_j & = & \begin{cases} b(8k+1) & \text{for } j = 1, 2, ...k \\ b(8k+1) & \text{for } j = (k+1), ..., 2k \\ b(6k-1) & \text{for } j = (2k+1), ..., 4k \end{cases} &
\end{array}
$$

We shall show that the problem of checking whether there exists a schedule with $\sum U_j \leq k$ is equivalent to the partition problem. First, observe that any job processed in the second batch or later will have a completion time of at least $\min\{2s_1 + s_2, s_1 + 2s_2\} = 6kb$. Therefore, all on-time jobs must be processed in the first batch.

Let $\mathbf{J} = \{j : 1 \leq j \leq 2k$ and job j is in the first batch$\}$. The completion time of every job is at least $s_1 + s_2 + \min_i \sum(t_{ij} : j \in \mathbf{J}) = 4kb + \sum(t_{1j} : j \in \mathbf{J}) = 4kb + 2|\mathbf{J}|b - \sum(a_j : 1 \leq j \leq k$ and $j \in \mathbf{J}) \geq 4kb + 2|\mathbf{J}|b - 2b$. Hence, if $|\mathbf{J}| \geq (k+1)$, then the completion time of every job is at least $6kb$ and, therefore, none of the last $2k$ jobs can be processed before their due dates. If none of the last $2k$ jobs are processed before their due dates, $\sum U_j \not\leq k$. Hence, we must have $|\mathbf{J}| \leq k$. Again, $\sum U_j \leq k$ and $n = 4k$ imply $|\mathbf{J}| \geq k$. Hence, in order to get $\sum U_j \leq k$, exactly k jobs from the first $2k$ jobs, and all jobs from the last $2k$ jobs must be processed in the first batch and completed before their due dates. Notice also that since the last $2k$ jobs are completed before their due dates, the first batch must start on machine 1.

In order to complete any of the last $2k$ jobs on time, the start time of the second machine, which is $6kb - \sum(a_j : 1 \leq j \leq k$ and $j \in \mathbf{J})$, must be less than or equal to $b(6k - 1)$. Again, in order to complete all jobs in $\mathbf{J}$ on time, the completion time of the first batch, which is $8kb + \sum(a_j : 1 \leq j \leq k$ and $j \in \mathbf{J})$, must be less than or equal to $b(8k + 1)$. Hence, $\sum(a_j : 1 \leq j \leq k$ and $j \in \mathbf{J}) = b$.

Thus, there exists a schedule with $\sum U_j \leq k$ if and only if there exists $\mathbf{B} \subset \mathbf{A}$

such that $\sum_{a_l \in \mathbf{B}} a_l = b$. ■

Theorem 6.2 *For both problems* $1O2|s_i|\sum w_jU_j$ *and* $1F2|s_i|\sum w_jU_j$ *there exists an optimal schedule which is a batching schedule and in which (i) all jobs in a batch are either early or tardy, (ii) all tardy jobs are processed in a single batch after the completion of all early batches, and (iii) the early jobs are processed according to the Earliest Due Date (EDD) rule.*

Proof: Consider a schedule σ, remove the tardy jobs and perform the left-shift operation to obtain the semi-active schedule σ' containing the remaining jobs. The schedule σ' does not contain any tardy job. It follows from Theorems 3.6 and 4.1 that if σ' is not a batching schedule in which jobs are processed in EDD order, a batching schedule can be obtained from σ' in which jobs are arranged in the EDD order and the maximum lateness does not increase. Consider any such schedule σ''. Since the maximum lateness does not increase in σ'', no job in schedule σ'' is tardy. Now, append a new batch containing all the jobs which are tardy in σ. The resulting schedule satisfies the condition stated in the Theorem and has weighted number of tardy jobs no more than in σ. ■

Henceforth, we shall assume that jobs are labelled so that $d_1 < d_2 < ... < d_n$. Although in the above we discuss the problem of minimizing the weighted number of tardy jobs, it will be convenient for us to consider the equivalent problem of maximizing the weighted number of early jobs. Henceforth, we shall consider the equivalent

problem of maximizing the weighted number of early jobs, $\sum w_j(1 - U_j)$.

6.2 Open Shop Problem

In this section we shall present two pseudo-polynomial dynamic programming recursions, O_1 and O_2 for the open shop problem. Algorithm O_1 is an extension of the algorithm presented by Hochbaum and Landy [58] and algorithm O_2 is an extension of the algorithm presented by Brucker and Kovalyov [24]. Note that both Hochbaum and Landy [58] and Brucker and Kovalyov [24] consider the problem $1|s_1, F = 1|\sum w_j U_j$ for which there exists a polynomial-time algorithm if all $w_j = 1$. On the other hand, our problem is $\mathcal{NP}$-hard even if all $w_j = 1$.

Both algorithms O_1 and O_2 schedule jobs in order of increasing due dates. Consider a partial schedule of jobs 1, ..., $j - 1$. We can add job j to the given partial schedule, in one of three ways:

1. Job j can be a tardy job.

2. Job j can be added to the last early batch.

3. Job j can be the only job in a new early batch.

Suppose that the operator spends an idle time, $\hat{t}$ after completing all the operations corresponding to the early jobs on one machine i_1 and before starting the immediate setup operation on the other machine $i_2 \neq i_1$. If $t_{i_1 j} \leq \hat{t}$, then the j-th job can be

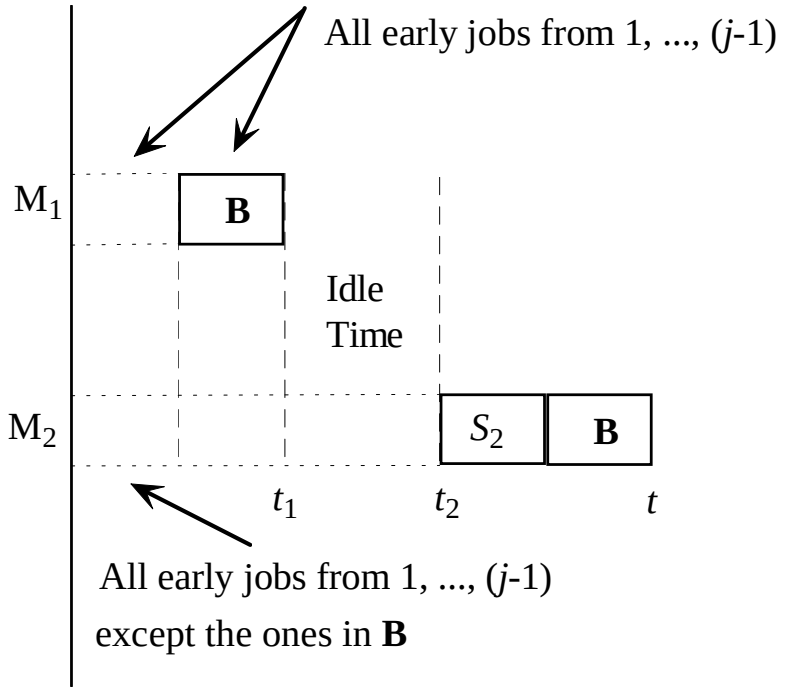

Figure 6.1: A partial schedule in which the last early batch is first processed on machine 1

inserted in the last early batch without causing any delay to the previously scheduled operations.

Each algorithm O_1 and O_2 addresses the following **question** at the j-th stage: given jobs 1, ..., j, does there exist a feasible schedule that has completion time equal to t for the last early job, weighted number of early jobs equal to w and idle time $\hat{t}$ after the completion of all the operations corresponding to the early jobs on one machine i_1 and before the start of the immediate setup operation on the other machine $i_2 \neq i_1$.

Let us clarify the question using an example. In Figure 6.1 we show a schedule

with jobs 1, ..., $(j-1)$. The last early batch, **B** is completed on machine 1 at time t_1. The operator is idle from time t_1 to time t_2. At time t_2 a setup on machine 2 starts and the setup is immediately followed by operations corresponding to batch **B** on machine 2. A job j' is processed on machines 1 and 2 before time t if and only if job j' is an early job. All the tardy jobs are processed on machines 1 and 2 in a single batch some time after time t.

Thus, if there exists a feasible schedule of the type shown in Figure 6.1 with weighted number of early jobs w, then we have an answer "yes" to the above question with $(j-1)$ jobs, idle time $\hat{t} = t_2 - t_1$, completion time t of the last early job, weighted number of early jobs w, and $i_1 = 1$.

Now, let us see how a "yes" answer from stage $(j-1)$ implies a "yes" answer at stage j. Suppose that there exists a feasible schedule of the type shown in Figure 6.1 with weighted number of early jobs w.

1. Since job j can be scheduled as a tardy job, we get a "yes" answer to the above question with j jobs, idle time $\hat{t} = t_2 - t_1$, completion time t of the last early job, weighted number of early jobs w, and $i_1 = 1$.

2. Job j can be added to the last early batch **B** only if the idle time $\hat{t} = t_2 - t_1 \geq t_{1j}$ and due date $d_j \leq t + t_{2j}$. If job j can be added to the last early batch **B**, we get that in the resulting schedule (see Figure 6.2) idle time is $(\hat{t} - t_{1j})$, completion time of the last early job is $(t + t_{2j})$ and weighted number of early

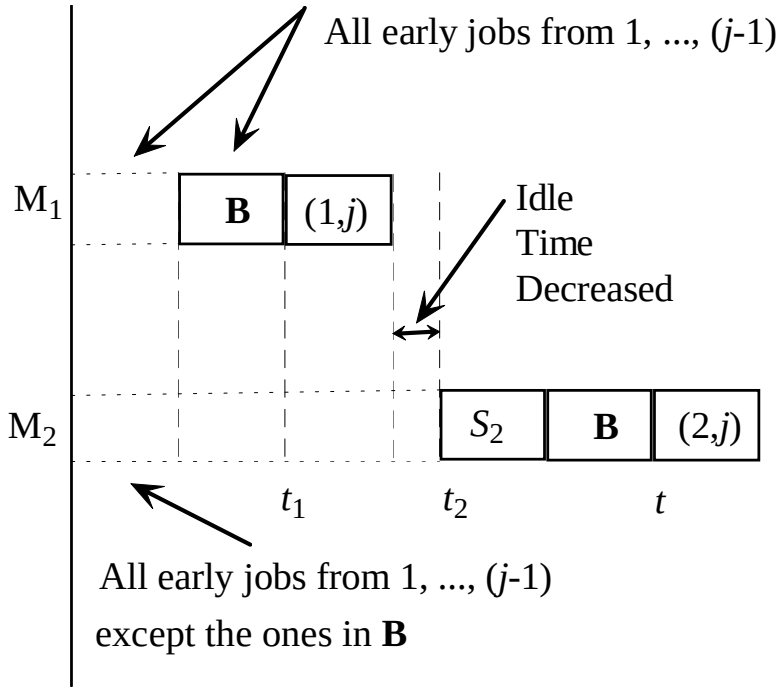

Figure 6.2: Job j is added to the last early batch

jobs is $(w+w_j)$. Hence, we get a "yes" answer to the above question with j jobs, idle time $(\hat{t}-t_{1j})$, the completion time $(t+t_{2j})$ of the last early job, weighted number of early jobs $(w+w_j)$, and $i_1=1$.

3. If job j is added as the only job in a new early batch, an idle time in the given schedule is not necessary. Hence, assume that there is no idle time in the given schedule. Operation $(2,j)$ starts immediately at time t, the operator spends an idle time t' as desired, the setup on machine 1 starts at time $(t+t')$ and the setup is immediately followed by operation $(1,j)$. The resulting schedule is shown in Figure 6.3. Since we require that job j be early, this way of adding job j is valid only if the completion time $(t+t'+s_1+t_{1j}+t_{2j})$ of job j is less than or equal to d_j. If the condition is satisfied, we get a "yes" answer to the above question with j jobs, idle time t', completion time $(t+t'+s_1+t_{1j}+t_{2j})$ of the last early job, weighted number of early jobs $(w+w_j)$, and $i_1=2$.

In the above, we outline how a feasible schedule at stage $(j-1)$ can be used to obtain a feasible schedule at the j-th stage. Algorithms O_1 and O_2 are based on this concept.

In algorithm O_1 the state variables are $\hat{t}$, t and i_1. If for some j, $\hat{t}$, t and i_1 we have w^* the largest value of w such that the answer to the above question is "yes", then we set

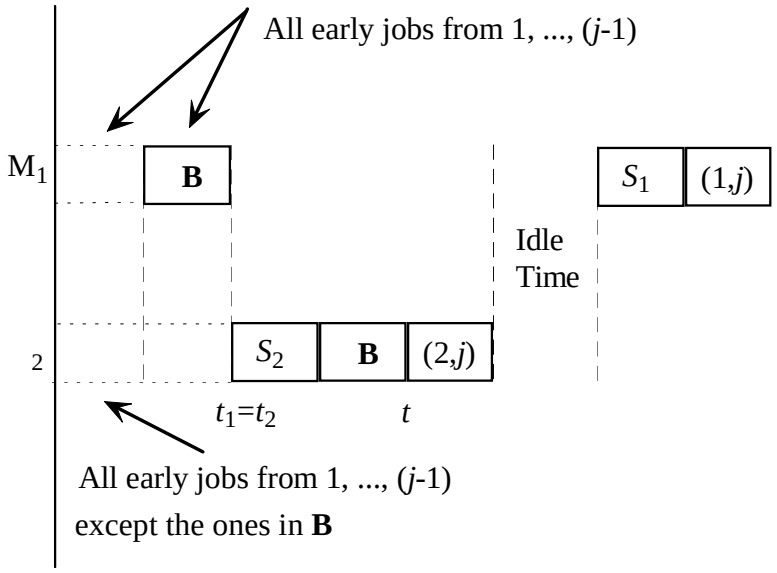

Figure 6.3: Job j is the first job of a new early batch

$$\begin{aligned} f_j(\hat{t}, t) &= w^* \quad \text{if } i_1 = 1 \\ g_j(\hat{t}, t) &= w^* \quad \text{if } i_1 = 2 \end{aligned}$$

In algorithm O_2 the state variables are $\hat{t}$, w and i_1. If for some j, $\hat{t}$, w and i_1 we have t^* the least value of t such that the answer to the above question is "yes", then we set

$$\begin{aligned} f_j(\hat{t}, w) &= t^* \quad \text{if } i_1 = 1 \\ g_j(\hat{t}, w) &= t^* \quad \text{if } i_1 = 2 \end{aligned}$$

It is clear by now that we need the state variable $\hat{t}$, the idle time of the operator, to facilitate checking whether a job j can be added to the last early batch of a partial schedule of jobs 1,...,$j-1$. We do not need to consider $\hat{t} > d_{\max}$ because jobs requiring $\hat{t} > d_{\max}$ can only be added to the partial schedule as a tardy job. Before adding a job to the last early batch that starts on machine 1, we check whether

$\hat{t} \geq t_{1j}$. Hence we do not need to consider $\hat{t} > \sum t_{1j}$. Thus, it suffices to consider $\hat{t} = 0, ..., \min\{\sum t_{1j}, d_{\max}\}$ while computing $f_j(\hat{t}, t)$ or $f_j(\hat{t}, w)$. Similarly, it suffices to consider $\hat{t} = 0, ..., \min\{\sum t_{2j}, d_{\max}\}$ while computing $g_j(\hat{t}, t)$ or $g_j(\hat{t}, w)$.

The idea of scheduling a job in one of the above three ways is due to Hochbaum and Landy [58]. Later on, Brucker and Kovalyov [24] also use the above three ways of scheduling a job. A key difference between our algorithms and the algorithms of Hochbaum and Landy [58] and Brucker and Kovalyov [24] is that our algorithms have a new state variable $\hat{t}$, the idle time between completion of all operations corresponding to the early jobs on one machine and the start of the immediate setup operation on the other machine. Furthermore, our algorithms do not use a state variable corresponding to the earliest due date of the last early batch, which is common to both the algorithms developed by Hochbaum and Landy [58] and Brucker and Kovalyov [24].

6.2.1 Algorithm O_1

For $j = 1, ..., n$ and $t = 0, 1, ..., \min\{\sum(t_{1j} + t_{2j}) + n(s_1 + s_2), d_{\max}\}$, let

$f_j(\hat{t}, t)$ = maximum weighted number of early jobs when jobs 1,...,j are scheduled, the operator processes the last early batch first on machine 1, is idle for a time $\hat{t}$ after completion of all operations on machine 1 and before starting the setup operation on machine 2, and completes all operations of early jobs on machine 2 at time t, where $\hat{t} = 0, 1, ..., \min\{\sum t_{1j}, d_{\max}\}$; and

$g_j(\hat{t},t)$ = maximum weighted number of early jobs when jobs 1,...,j are scheduled, the operator processes the last early batch first on machine 2, is idle for a time $\hat{t}$ after completion of all operations on machine 2 and before starting the setup operation on machine 1, and completes all operations of early jobs on machine 1 at time t, where $\hat{t} = 0, 1, ..., \min\{\sum t_{2j}, d_{\max}\}$.

Introduce a dummy job 0 and initialize: $f_0(\hat{t},t) = 0$ and $g_0(\hat{t},t) = 0$ for all $\hat{t},t$. Compute the quantities $f_j(\hat{t},t) = \max\{f_{j-1}(\hat{t},t), \theta_1, \theta_2, \theta_3\}$ and $g_j(\hat{t},t) = \max\{g_{j-1}(\hat{t},t), \theta'_1, \theta'_2, \theta'_3\}$, for all $j,\hat{t},t$, where

$$\theta_1 = \begin{cases} w_j + f_{j-1}(\hat{t}+t_{1j}, t-t_{2j}) & \text{if } f_{j-1}(\hat{t}+t_{1j}, t-t_{2j}) > 0 \\ & \text{and } t_{2j} \le t \le d_j \\ -\infty & \text{otherwise} \end{cases}$$

$$\theta_2 = \begin{cases} w_j + g_{j-1}(0, t-s_2-t_{1j}-t_{2j}-\hat{t}) & \text{if } g_{j-1}(0, t-s_2-t_{1j}-t_{2j}-\hat{t}) > 0 \\ & \text{and } s_2+t_{1j}+t_{2j}+\hat{t} \le t \le d_j \\ -\infty & \text{otherwise} \end{cases}$$

$$\theta_3 = \begin{cases} w_j & \text{if } t = s_1+s_2+t_{1j}+t_{2j}+\hat{t} \\ & \text{and } t \le d_j \\ -\infty & \text{otherwise} \end{cases}$$

and

$$\theta'_1 = \begin{cases} w_j + g_{j-1}(\hat{t}+t_{2j}, t-t_{1j}) & \text{if } g_{j-1}(\hat{t}+t_{2j}, t-t_{1j}) > 0 \\ & \text{and } t_{1j} \le t \le d_j \\ -\infty & \text{otherwise} \end{cases}$$

$$\theta'_2 = \begin{cases} w_j + f_{j-1}(0, t-s_1-t_{1j}-t_{2j}-\hat{t}) & \text{if } f_{j-1}(0, t-s_1-t_{1j}-t_{2j}-\hat{t}) > 0 \\ & \text{and } s_1+t_{1j}+t_{2j}+\hat{t} \le t \le d_j \\ -\infty & \text{otherwise} \end{cases}$$

$$\theta'_3 = \begin{cases} w_j & \text{if } t = s_1+s_2+t_{1j}+t_{2j}+\hat{t} \\ & \text{and } t \le d_j \\ -\infty & \text{otherwise} \end{cases}$$

The maximum weighted number of early jobs is equal to $\max_t\{f_n(0,t), g_n(0,t)\}$.

The quantities θ_1 and θ_1' correspond to adding job j to the last early batch, θ_2 and θ_2' correspond to adding job j as the first job of an early batch which is preceded by at least one other early batch, and θ_3 and θ_3' correspond to adding job j as the first job of the first early batch. Setting $f_j(\hat{t},t) = f_{j-1}(\hat{t},t)$ and $g_j(\hat{t},t) = g_{j-1}(\hat{t},t)$ corresponds to adding job j as a tardy job.

For $T_1 = \min\{\max\{\sum t_{1j}, \sum t_{2j}\}, d_{\max}\}$ and $T_2 = \min\{\sum(t_{1j} + t_{2j}) + n(s_1 + s_2), d_{\max}\}$, the worst case time complexity is $O(nT_1T_2)$. To compute $f_j(\hat{t},t)$ and $g_j(\hat{t},t)$ we need the values $f_{j-1}(\hat{t},t)$ and $g_{j-1}(\hat{t},t)$, but not the values $f_{j-2}(\hat{t},t)$, ..., $f_0(\hat{t},t)$ and $g_{j-2}(\hat{t},t)$, ..., $g_0(\hat{t},t)$. Hence, the space requirement is $O(T_1T_2)$. This gives:

Theorem 6.3 *Algorithm O_1 solves the open shop problem with running time $O(nT_1T_2)$ and space requirement $O(T_1T_2)$.*

6.2.2 Algorithm O_2

For $j = 1, ..., n$ and $w = 0, 1, ..., \sum w_j$, let

$f_j(\hat{t}, w)$ = minimum completion time of the last early job processed when jobs 1,...,j are scheduled, the operator processes the last early batch first on machine 1, is idle for a time $\hat{t}$ after completion of all operations on machine 1 and before starting the setup operation on machine 2, and attains a weighted number of early jobs w, where $\hat{t} = 0, 1, ..., \min\{\sum t_{1j}, d_{\max}\}$; and

$g_j(\hat{t}, w)$ = minimum completion time of the last early job processed when jobs 1,...,j are scheduled, the operator processes the last early batch first on machine 2, is idle for a time $\hat{t}$ after completion of all operations on machine 2 and before starting the setup operation of machine 1, and attains a weighted number of early jobs w, where $\hat{t} = 0, 1, ..., \min\{\sum t_{2j}, d_{\max}\}$.

Introduce a dummy job 0 and initialize: $f_0(\hat{t}, w) = \infty$ and $g_0(\hat{t}, w) = \infty$ for all $\hat{t}, w$. Compute the quantities $f_j(\hat{t}, w) = \min\{f_{j-1}(\hat{t}, w), \theta_1, \theta_2, \theta_3\}$ and $g_j(\hat{t}, w) = \min\{g_{j-1}(\hat{t}, w), \theta'_1, \theta'_2, \theta'_3\}$, for all $j, \hat{t}, w$, where

$$\theta_1 = \begin{cases} f_{j-1}(\hat{t} + t_{1j}, w - w_j) + t_{2j} & \text{if } f_{j-1}(\hat{t} + t_{1j}, w - w_j) + t_{2j} \le d_j \\ & \text{and } w > w_j \\ \infty & \text{otherwise} \end{cases}$$

$$\theta_2 = \begin{cases} s_2 + t_{1j} + t_{2j} + \hat{t} + g_{j-1}(0, w - w_j) & \text{if } s_2 + t_{1j} + t_{2j} + \hat{t} \\ & + g_{j-1}(0, w - w_j) \le d_j \\ & \text{and } w > w_j \\ \infty & \text{otherwise} \end{cases}$$

$$\theta_3 = \begin{cases} s_1 + s_2 + t_{1j} + t_{2j} + \hat{t} & \text{if } s_1 + s_2 + t_{1j} + t_{2j} + \hat{t} \le d_j \\ & \text{and } w = w_j \\ \infty & \text{otherwise} \end{cases}$$

and

$$\theta'_1 = \begin{cases} g_{j-1}(\hat{t} + t_{2j}, w - w_j) + t_{1j} & \text{if } g_{j-1}(\hat{t} + t_{2j}, w - w_j) + t_{1j} \le d_j \\ & \text{and } w > w_j \\ \infty & \text{otherwise} \end{cases}$$

$$\theta'_2 = \begin{cases} s_1 + t_{1j} + t_{2j} + \hat{t} + f_{j-1}(0, w - w_j) & \text{if } s_1 + t_{1j} + t_{2j} + \hat{t} \\ & + f_{j-1}(0, w - w_j) \le d_j \\ & \text{and } w > w_j \\ \infty & \text{otherwise} \end{cases}$$

$$\theta'_3 = \begin{cases} s_1 + s_2 + t_{1j} + t_{2j} + \hat{t} & \text{if } s_1 + s_2 + t_{1j} + t_{2j} + \hat{t} \le d_j \\ & \text{and } w = w_j \\ \infty & \text{otherwise} \end{cases}$$

The maximum weighted number of early jobs is equal to $\max\{w : f_n(0, w) < \infty$ or

$g_n(0,w) < \infty\}$. If $f_n(0,w) = g_n(0,w) = \infty$ for all w, then the maximum weighted number of early jobs is 0.

The quantities θ_1 and θ_1' correspond to adding job j to the last early batch, θ_2 and θ_2' correspond to adding job j as the first job of an early batch which is preceded by at least one other early batch, and θ_3 and θ_3' correspond to adding job j as the first job of the first early batch. Setting $f_j(\hat{t},w) = f_{j-1}(\hat{t},w)$ and $g_j(\hat{t},w) = g_{j-1}(\hat{t},w)$ corresponds to adding job j as a tardy job.

For $T_1 = \min\{\max\{\sum t_{1j}, \sum t_{2j}\}, d_{\max}\}$ and $W = \sum w_j$ the worst case time complexity is $O(nT_1W)$. To compute $f_j(\hat{t},w)$ and $g_j(\hat{t},w)$ we need the values $f_{j-1}(\hat{t},w)$ and $g_{j-1}(\hat{t},w)$, but not the values $f_{j-2}(\hat{t},w)$, ..., $f_0(\hat{t},w)$ and $g_{j-2}(\hat{t},w)$, ..., $g_0(\hat{t},w)$. Hence, the space requirement is $O(T_1W)$. This gives:

Theorem 6.4 *Algorithm O_2 solves the open shop problem with running time $O(nT_1W)$ and space requirement $O(T_1W)$.*

6.3 Flow Shop Problem

The flow shop problem can be solved similarly. For completeness, we present the algorithms below.

6.3.1 Algorithm F_1

For $j = 1, ..., n$, $\hat{t} = 0, 1, ..., \min\{\sum t_{1j}, d_{\max}\}$ and $t = 0, 1, ..., \min\{\sum(t_{1j} + t_{2j}) + n(s_1 + s_2), d_{\max}\}$, let

$f_j(\hat{t},t)$ = maximum weighted number of early jobs when jobs 1,...,j are scheduled, the operator processes the last early batch first on machine 1, is idle for a time $\hat{t}$ after completion of all operations on machine 1 and before starting the setup operation on machine 2, and completes all operations of early jobs on machine 2 at time t.

Introduce a dummy job 0 and initialize: $f_0(\hat{t},t) = 0$ for all $\hat{t},t$. Compute for $j = 1,2,...,n$, $\hat{t} = 0,1,...,$ $\min\{\sum t_{1j},\ d_{\max}\}$ and $t = 0,1,...,$ $\min\{\sum(t_{1j} + t_{2j}) + n(s_1 + s_2),\ d_{\max}\}$ the quantity $f_j(\hat{t},t) = \max\{f_{j-1}(\hat{t},t), \theta_1, \theta_2\}$, where

$$\theta_1 = \begin{cases} w_j + f_{j-1}(\hat{t} + t_{1j}, t - t_{2j}) & \text{if } f_{j-1}(\hat{t} + t_{1j}, t - t_{2j}) > 0 \\ & \text{and } t_{2j} \le t \le d_j \\ -\infty & \text{otherwise} \end{cases}$$

$$\theta_2 = \begin{cases} w_j + f_{j-1}(0, t - s_1 - s_2 - t_{1j} - t_{2j} - \hat{t}) & \text{if } s_1 + s_2 + t_{1j} + t_{2j} \\ & + \hat{t} \le t \le d_j \\ -\infty & \text{otherwise} \end{cases}$$

The maximum weighted number of early jobs is equal to $\max_t\{f_n(0,t)\}$.

The quantity θ_1 corresponds to adding job j to the last early batch and θ_2 corresponds to adding job j as the first job of an early batch. Setting $f_j(\hat{t},t) = f_{j-1}(\hat{t},t)$ corresponds to adding job j as a tardy job.

For $T_1 = \min\{\sum t_{1j},\ d_{\max}\}$ and $T_2 = \min\{\sum(t_{1j} + t_{2j}) + n(s_1 + s_2),\ d_{\max}\}$, the worst case time complexity is $O(nT_1T_2)$. To compute $f_j(\hat{t},t)$ we need the values $f_{j-1}(\hat{t},t)$, but not the values $f_{j-2}(\hat{t},t)$, ..., $f_0(\hat{t},t)$. Hence, and the space requirement is $O(T_1T_2)$. This gives:

Theorem 6.5 *Algorithm F_1 solves the flow shop problem with running time $O(nT_1T_2)$ and space requirement $O(T_1T_2)$.*

6.3.2 Algorithm F_2

For $j = 1, ..., n$, $\hat{t} = 0, 1, ...,$ $\min\{\sum t_{1j},\ d_{\max}\}$ and $w = 0, 1, ...,$ $\sum w_j$, let

$f_j(\hat{t}, w)$ = minimum completion time of the last early job processed when jobs 1,...,j are scheduled, the operator processes the last early batch first on machine 1, is idle for a time $\hat{t}$ after completion of all operations on machine 1 and before starting the setup operation on machine 2, and attains a weighted number of early jobs w.

Introduce a dummy job 0 and initialize: $f_0(\hat{t}, w) = \infty$ for all $\hat{t}, w$. Compute for $j = 1, 2, ..., n$, $\hat{t} = 0, 1, ...,$ $\min\{\sum t_{1j}, d_{\max}\}$ and $w = 0, 1, ...,$ $\sum w_j$ the quantity $f_j(\hat{t}, w) = \min\{f_{j-1}(\hat{t}, w),\ \theta_1,\ \theta_2,\ \theta_3\}$, where

$$\theta_1 = \begin{cases} f_{j-1}(\hat{t} + t_{1j}, w - w_j) + t_{2j} & \text{if } f_{j-1}(\hat{t} + t_{1j}, w - w_j) + t_{2j} \le d_j \text{ and } w > w_j \\ \infty & \text{otherwise} \end{cases}$$

$$\theta_2 = \begin{cases} s_1 + s_2 + t_{1j} + t_{2j} + \hat{t} + f_{j-1}(0, w - w_j) & \text{if } s_1 + s_2 + t_{1j} + t_{2j} + \hat{t} + f_{j-1}(0, w - w_j) \le d_j \text{ and } w > w_j \\ \infty & \text{otherwise} \end{cases}$$

$$\theta_3 = \begin{cases} s_1 + s_2 + t_{1j} + t_{2j} + \hat{t} & \text{if } s_1 + s_2 + t_{1j} + t_{2j} + \hat{t} \le d_j \text{ and } w = w_j \\ \infty & \text{otherwise} \end{cases}$$

The maximum weighted number of early jobs is equal to $\max\{w : f_n(0, w) < \infty\}$. If $f_n(0, w) = \infty$ for all w, then the maximum weighted number of early jobs is 0.

The quantity θ_1 corresponds to adding job j to the last early batch, θ_2 corresponds to adding job j as the first job of an early batch which is preceded by at least one other early batch, and θ_3 corresponds to adding job j as the first job of the first early

batch. Setting $f_j(\hat{t}, w) = f_{j-1}(\hat{t}, w)$ corresponds to adding job j as a tardy job.

For $T_1 = \min\{\sum t_{1j}, d_{\max}\}$ and $W = \sum w_j$ the worst case time complexity is $O(nT_1W)$. To compute $f_j(\hat{t}, w)$ we need the values $f_{j-1}(\hat{t}, w)$, but not the values $f_{j-2}(\hat{t}, w)$, ..., $f_0(\hat{t}, w)$. Hence, the space requirement is $O(T_1W)$. This gives:

Theorem 6.6 *Algorithm F_2 solves the flow shop problem with running time $O(nT_1W)$ and space requirement $O(T_1W)$.*

6.4 Computational Experience

The algorithms are tested on a randomly generated set of problems. Hall and Posner [56] discuss on generating experimental data for machine scheduling problems. They observe that most data generation methods generate the processing times independently from a uniform integer distribution $U[a, b]$, where $0 < a < b$. Usually, $a = 1$. Due dates are generated from a uniform integer distribution $U[a'h, b'h]$, where a' and b' come from a set of small values and h is a function of processing times, e.g., expected value of total processing time.

We consider problems with $n = 25, 50, 75$ and 100. For each n, 10 problems are generated. We generate $s_i \sim U[1, 4]$, $t_{ij} \sim U[1, 15]$, and $w_j \sim U[1, 25]$. We assume $d_j \geq s_1 + s_2 + t_{1j} + t_{2j}$, since otherwise job j will be tardy in any schedule, and can be eliminated from further consideration. The expected value of $\sum(t_{1j} + t_{2j})$ is $16n$. We generate d_j as $d_j \sim s_1 + s_2 + t_{1j} + t_{2j} + U[0, 16n]$.

	Algorithms			
n	O_1	O_2	F_1	F_2
25	10.9	9.3	4.8	3.9
50	82.6	69.0	41.7	31.7
75	297.1	255.8	143.6	119.6
100	685.2	602.7	309.0	264.4

Table 6.1: Average running times (in seconds) of algorithms for weighted tardiness

The programs are written in C and run on an IBM RISC 6000 43P M140 using the AIX operating system. The results are summarized in Table 6.1. The time required to solve the open shop problem is between 2 to 3 times that required to solve the flow shop problem. Algorithm O_2 is faster than algorithm O_1 and algorithm F_2 is faster than algorithm F_1.

6.5 Summary

There are two styles of algorithms for the single machine scheduling problems with weighted number of tardy jobs objective. One style is used by Hochbaum and Landy [58] and Lawler and Moore [77] and the other by Brucker and Kovalyov [24] and Sahni [104]. In this chapter we address the problems $1O2|s_i|\sum w_jU_j$ and $1F2|s_i|\sum w_jU_j$ using both the styles.

Both Brucker and Kovalyov [24] and Hochbaum and Landy [58] consider the problem $1|s_1, F=1|\sum w_jU_j$ in which jobs are processed in batches. Hochbaum and Landy [58] develop a procedure to construct a schedule with jobs 1, ..., j from a given par-

tial schedule with jobs 1, ..., $(j-1)$. Later, Brucker and Kovalyov [24] use the same schedule construction procedure.

We show in Theorem 6.2 that for both problems $1O2|s_i|\sum w_jU_j$ and $1F2|s_i|\sum w_jU_j$ there exists an optimal schedule which is a batching schedule and in which all early jobs are processed in the EDD order before all tardy jobs. Theorem 6.2 allows us to use the schedule construction procedure of Hochbaum and Landy [58].

However, the problems addressed by Brucker and Kovalyov [24], Hochbaum and Landy [58], Lawler and Moore [77] and Sahni [104] are polynomially solvable if all $w_j = 1$. In contrast, as we show in Theorem 6.1, our problems are $\mathcal{NP}$-hard even if all $w_j = 1$.

Still, we extend the dynamic programming algorithms developed by Hochbaum and Landy [58] and Brucker and Kovalyov [24]. A key difference between our algorithms and the algorithms of Hochbaum and Landy [58] and Brucker and Kovalyov [24] is that our algorithms have a new state variable $\hat{t}$, which is an idle time of the operator. Furthermore, our algorithms do not use a state variable corresponding to the earliest due date of the last early batch, which is common to both the algorithms developed by Hochbaum and Landy [58] and Brucker and Kovalyov [24].

We test all the algorithms on a randomly generated set of problems.

In Table 6.2, we summarize the results on various cases of the (weighted) number of tardy jobs objective.

Problem	**Result (Chapter 6)**	**Previously Known Result**
$1O2\|s_i\|\sum U_j$	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard (Gerodimos et al. [45, 46])
$1F2\|s_i\|\sum U_j$	$\mathcal{NP}$-hard	—
$1O2\|s_i\|\sum w_jU_j$	$O(n\min\{\max\{\sum t_{1j}, \sum t_{2j}\}, d_{\max}\}\min\{\sum(t_{1j}+t_{2j})+n(s_1+s_2), d_{\max}\})$	$O(nd_{\max}^2)$ (Gerodimos et al. [45, 46])
	$O(n\min\{\max\{\sum t_{1j}, \sum t_{2j}\}, d_{\max}\}\sum w_j)$	—
$1F2\|s_i\|\sum w_jU_j$	$O(n\min\{\sum t_{1j}, d_{\max}\}\min\{\sum(t_{1j}+t_{2j})+n(s_1+s_2), d_{\max}\})$	—
	$O(n\min\{\sum t_{1j}, d_{\max}\}\sum w_j)$	—

Table 6.2: Summary of results on two-machine cases with the weighted number of tardy jobs objective

Chapter 7

The Total Weighted Completion Time Objective

In this chapter we shall consider the problems with total weighted completion time objective. This objective is the most obscure among all the objectives we study. As we have discussed in Section 3.5 the problem $1|s_1, F = 1|\sum w_j C_j$ is a special case of the problem $1F2|s_i|\sum w_j C_j$ and Albers and Brucker [4] show that the problem $1|s_1, F = 1|\sum w_j C_j$ is strongly $\mathcal{NP}$-hard. Hence, the problem $1F2|s_i|\sum w_j C_j$ is strongly $\mathcal{NP}$-hard.

However, the problem $1F2|s_i|\sum C_j$ is open. As we have discussed in Section 1.4.3, Ding [34] considers the problem of scheduling products with common and unique components as introduced by Baker [12]. Ding [34] maintains the assumption of batch availability . However, unlike Baker, Ding does not restrict the problem to the case of *agreeable processing time*. Ding's problem is equivalent to the problem $1F2|s_i|\sum C_j$ in which the first machine represents the production of the common components

and the second machine represents the production of the unique components. Ding proposes a heuristic procedure.

The problem $1O2|s_i|\sum C_j$ is also open. As we have discussed in Section 1.4.4, Julien and Magazine [64] and Julien [63] discuss the customer order scheduling problem. The customer order scheduling problem with two product types and each customer requiring at least one unit of each product can be modelled as a problem $1O2|s_i|\sum C_j$. Julien and Magazine [64] and Julien [63] discuss some dominance properties, special cases, relative performances of various types of schedules and some heuristic and lower bounding procedures.

In this chapter, we revisit some of the flow shop and open shop fixed-sequence cases. One motivation for analyzing the fixed-sequence cases is that it is sometimes possible to obtain a job-order which dominates all the other job-orders. For example, it follows from Theorem 4.2 that one such case is that of *agreeable processing time and weight*. Another motivation is to eventually obtain an enumeration scheme. However, as there are $n!$ job-orders, we seek an alternate scheme.

Such an alternate scheme is to enumerate over all batching policies . There are 2^{n-1} batching policies . It turns out that both the flow shop and open shop problems with the objective $\sum w_jC_j$ are $\mathcal{NP}$-hard even if the batching policy is fixed. However, the cases with $w_j = 1$ are efficiently solvable.

In the next chapter, we develop a branch and bound algorithm for the problem

$1F2|s_i|\sum C_j$ using the results on the flow shop fixed-sequence and fixed batching policy cases with the objective $\sum C_j$.

As we have discussed in Section 2.1.2, Coffman et al. [31] show that Baker's problem of scheduling products with common and unique components and batch availability reduces to the problem $1|s_1, F = 1|\sum C_j$. As Baker assumes *agreeable processing time*, the shortest processing time order dominates all the other job-orders. Coffman et al.'s assertion that Baker's problem reduces to the problem $1|s_1, F = 1|\sum C_j$ is based on the fact that for Baker's problem we can fix the sequence without any loss of optimality. Using this fact, Coffman et al. [31] present an algorithm for the problem $1|s_1, F = 1|\sum C_j$ that runs in $O(n)$ time after sorting the jobs. Albers and Brucker [4] generalize the algorithm of Coffman et al. [31] to the problem $1|s_1, F = 1|\sum w_jC_j$ and present an algorithm which runs in $O(n)$ time after sorting the jobs. Using arguments similar to the ones used by Coffman et al., we can show that the problem $1F2|s_i$,*fixed sequence*$|\sum w_jC_j$ reduces to $1|s_1, F = 1$,*fixed sequence*$|\sum w_jC_j$. This means that the problem $1F2|s_i$,*fixed sequence*$|\sum w_jC_j$ can be solved in $O(n)$ time using the algorithm of Albers and Brucker [4].

However, a similar reduction of the problem $1O2|s_i$,*fixed sequence*$|\sum w_jC_j$ to the problem $1|s_1, F = 1$,*fixed sequence*$|\sum w_jC_j$ is not known. Each of the following problems are special cases of the problem $1O2|s_i$,*fixed sequence*$|\sum C_j$: (i) the customer order scheduling problem with a fixed sequence, two product types and each

customer requiring at least one unit of each product; (ii) the problem of scheduling products with common and unique components with item availability and agreeable processing times; and (iii) the problem $1|s_i$, *two operation, aptw*$|\sum C_j$. For problem (i) Julien and Magazine [64] and Julien [63] present an $O(n^2)$ time algorithm. For problem (ii) Sung and Park [111] present two $O(n^2)$ time algorithms. For problem (iii) Gerodimos et al. [44] present an $O(n^2)$ time algorithm. However, in this chapter we shall show that the problem further generalized to the weighted case, $1O2|s_i$,*fixed sequence*$|\sum w_jC_j$ is solvable in $O(n)$ time.

Based on the development on what are called *Monge-array algorithms,* we show that each problem $1F2|s_i$,*fixed sequence*$|\sum w_jC_j$ and $1O2|s_i$,*fixed sequence*$|\sum w_jC_j$ can be solved using a similar approach.

An $u \times v$ array $\mathbf{A} = \{a[j, j']\}$ is said to satisfy the *Monge property* if for any $1 \leq j < u$ and $1 \leq j' < v$, we have $a[j, j' +1] + a[j +1, j'] - a[j, j'] - a[j +1, j' +1] \geq 0$. Aggarwal et al. [2] show that we can find all row or column minima of such a matrix $\mathbf{A}$ in $O(u + v)$ time, provided that each entry of $\mathbf{A}$ can be obtained in constant time. Wilber [122] extends the algorithm of Aggarwal et al. to the context of dynamic programming. Wilber considers the following problem: given an array $\mathbf{A} = \{a[j, j'] : 1 \leq j \leq j' \leq u\}$ that satisfies the Monge property , and given $f(u+1)$,

compute

$$f(j) = \min_{j \leq j' \leq u} \{a[j, j'] + h(j' + 1)\} \; \forall 1 \leq j \leq u, \tag{7.1}$$

where $h(j' + 1)$ is computed from $f(j' + 1)$ in constant time. We are interested in the special case in which $h(j' + 1) = f(j' + 1)$. Hence, the condition that $h(j' + 1)$ be computed from $f(j' + 1)$ in constant time is satisfied. Wilber shows that $f(j)$, $\forall 1 \leq j \leq u$ can be computed in $O(u)$ time. Eppstein [37] extended Wilber's algorithm for interleaved computation. Eppstein's algorithm allows the computation of $h(2)$, ..., $h(u + 1)$ to be interleaved with the computation of some other sequence $h'(2)$, ..., $h'(u + 1)$ such that $h(j)$ depends on $h'(j + 1)$, ..., $h'(u + 1)$ and $h'(j)$ depends on $h(j + 1)$, ..., $h(u + 1)$. Eppstein's algorithm also requires $O(u)$ time. Galil and Park [40] further generalize and simplifies Eppstein's algorithm.

We shall show that the problem $1F2|s_i$,*fixed sequence*$|\sum w_j C_j$ can be solved by recursively solving an equation of the type Equation 7.1. The problem $1F2|s_i$,*fixed sequence*$|\sum w_j C_j$ calls for an interleaved computation, because we get two equations of the type Equation 7.1.

7.1 Contribution of Operation and Setup to $\sum w_j C_j$

We can define the "contribution" of each operation and setup so that the total weighted completion time is obtained by summing up the contributions of all operations and setups.

For any schedule σ and for each operation or setup T, let $\mathbf{J}_T(\sigma) = \{j$: either T is the last operation of job j or the last operation of job j is processed after $T\}$. Similarly, for any schedule σ and for each job j, let $\mathbf{T}_j(\sigma) = \{T$: either T is the last operation of job j or the last operation of job j is processed after $T\}$. Observe that $j \in \mathbf{J}_T(\sigma)$ if and only if $T \in \mathbf{T}_j(\sigma)$.

Definition 7.1 *For each operation or setup T let $t(T)$ be the processing time of T. For any schedule σ the* contribution of T *is $\eta(T) = t(T) \sum_{j \in \mathbf{J}_T(\sigma)} w_j$.*

Remark 7.1 *For any schedule σ the total weighted completion time is $\sum \eta(T)$.*

Proof: For each job j, the completion time C_j of job j is given as $C_j = \sum_{T \in \mathbf{T}_j(\sigma)} t(T)$. The total weighted completion time is $\sum w_j C_j = \sum w_j \sum_{T \in \mathbf{T}_j(\sigma)} t(T)$. Consider any job j and an operation or setup T. The term $w_j t(T)$ appears at most once in each of the expressions in $\sum w_j C_j$ and $\sum \eta(T)$. Since $j \in \mathbf{J}_T(\sigma)$ if and only if $T \in \mathbf{T}_j(\sigma)$, the term $w_j t(T)$ appears in the expression $\sum \eta(T)$ if and only if the term $w_j t(T)$ appears in the expression $\sum w_j C_j$. Hence, $\sum \eta(T) = \sum w_j C_j$. ■

7.2 The Fixed-Sequence Case Revisited

In this section we shall improve the time complexity of some of the algorithms we discussed in Chapter 4. We shall show that problem $1Fm|s_i$,*fixed sequence*$|\sum w_j C_j$ with $m \geq 3$ can be solved in time $O(mn^3)$, and problems $1F2|s_i$,*fixed sequence*$|\sum w_j C_j$ and $1O2|s_i$,*fixed sequence*$|\sum w_j C_j$ can be solved in time $O(n)$.

Since the sequence is fixed and known, we may assume, by relabelling if necessary, that the sequence is 1, 2, ..., n. Let W_j be the total weight of jobs j, j+1, ..., n and $\tau_{jj'i}$ be the total processing time of jobs j,j+1,..., j' on machine i. Hence, $W_j = \sum_{u=j}^{n} w_u$ and $\tau_{jj'i} = \sum_{u=j}^{j'} t_{iu}$.

Notice that if operation (i, j) is the last operation of job j, then the contribution of (i, j) to the total weighted completion time is $W_j t_{mj}$. If job j is the first job completed after operation (i, j'), then the contribution of (i, j') to the total weighted completion time is $W_j t_{ij'}$. If job j is the first job completed after a setup from machine i to i', then the contribution of the setup to the total weighted completion time is $s_{ii'} W_j$.

7.2.1 The m-machine Flow Shop Case

In this section we shall consider the problem $1Fm|s_i, \textit{fixed sequence}|\sum w_j C_j$. Throughout this section, $\sigma_{j,j',i}$ denotes a partial schedule of jobs j, ..., j' on machines i, ..., m, given that job j starts post-setup processing at time zero on machine i.

Let $h_{j'}(j, i)$ be the optimal total weighted completion time of jobs j, ..., n over all schedules of type $\sigma_{j,n,i}$ given that job j' is the largest indexed job such that jobs j, ..., j' are processed in a single setup on machine i.

Let $h(j,\ i)$ be the optimal total weighted completion time of jobs j, ..., n over all schedules of type $\sigma_{j,n,i}$. Notice that $h(j,\ i) = \min_{j \le j' \le n}\{h_{j'}(j, i)\}$. It follows from Remark 4.1 that the optimal total weighted completion time is $h^* = s_1 W_1 + h(1, 1)$.

For any partial schedule $\sigma_{j,j',i}$, define the following quantities:

$C_u(\sigma_{j,j',i})$ is the completion time of job u processed in $\sigma_{j,j',i}$, and

$$f(\sigma_{j,j',i}) = \begin{cases} \sum_{u=j}^{j'} w_u C_u(\sigma_{j,j',i}) + W_{j'+1} C_{j'}(\sigma_{j,j',i}) & \text{if } j' < n \\ \sum_{u=j}^{n} w_u C_u(\sigma_{j,n,i}) & \text{if } j' = n. \end{cases}$$

Let $f_{j''}(j,j',i) = \min\{f(\sigma_{j,j',i}) : \sigma_{j,j',i}$ is a schedule of jobs j, ..., j' on machines i, ..., m given that job j starts post-setup processing at time zero on machine i and job j'' is the largest job such that jobs $j, ..., j''$ are processed in a single setup on machine $i\}$. Finally, let $f^*(j,j',i) = \min_{j \le j'' \le j'}\{f_{j''}(j,j',i)\}$.

Now, we shall interpret $f(\sigma_{j,j',i})$, $f_{j''}(j,j',i)$ and $f^*(j,j',i)$. The quantity $f(\sigma_{j,j',i})$ represents the contribution of all the operations and setups performed in the partial schedule $\sigma_{j,j',i}$ to the total weighted completion time of jobs j, ..., n.

The quantity $f_{j''}(j,j',i)$ represents the minimum contribution of operations in $\mathbf{O}$ $=\{(\hat{\imath},\hat{\jmath}) : i \le \hat{\imath} \le m$ and $j \le \hat{\jmath} \le j'\}$ and all setups required to process operations in $\mathbf{O}$ to the total weighted completion time of jobs j, ..., n if (i) processing of operations in $\mathbf{O}$ is started when machine i is current and no operation $(\hat{\imath},\hat{\jmath}) \notin \mathbf{O}$ is processed between the start and completion of operations in $\mathbf{O}$; (ii) job j'' is the largest job such that jobs $j, ..., j''$ are processed in a single setup on machine i; and (iii) either $j' = n$ or jobs j' and $(j'+1)$ are processed in two different setups on machine i.

The quantity $f^*(j,j',i)$ represents the minimum contribution of operations in $\mathbf{O}$ $=\{(\hat{\imath},\hat{\jmath}) : i \le \hat{\imath} \le m$ and $j \le \hat{\jmath} \le j'\}$, and all setups required to process operations in $\mathbf{O}$, to the total weighted completion time of jobs j, ..., n if (i) processing of operations in $\mathbf{O}$ is started when machine i is current and no operation $(\hat{\imath},\hat{\jmath}) \notin \mathbf{O}$ is processed

between the start and completion of operations in **O**; and (ii) either $j' = n$ or jobs j' and $(j' + 1)$ are processed in two different setups on machine i.

In the following we shall discuss the computation of $f^*(j, j', i)$ and $h(j, i)$ separately.

Computation of $f^*(j, j', i)$

Let us discuss the computation of $f^*(j, j', i)$.

Case 1: $i = m$. The quantity $f(\sigma_{j,j',m})$ is minimized by processing jobs j, ..., j' on machine m without any setup. If $j' < n$, then $f^*(j, j', m) = \sum_{u=j}^{j'} w_u \sum_{v=j}^{u} t_{mv}$ $+W_{j'+1} \sum_{v=j}^{j'} t_{mv} = \sum_{u=j}^{j'} W_u t_{mu}$. If $j' = n$, then $f^*(j, j', m) = \sum_{u=j}^{n} w_u \sum_{v=j}^{u} t_{mv}$ $= \sum_{u=j}^{n} W_u t_{mu}$. Hence, for all $j' \leq n$, we have

$$f^*(j, j', m) = \sum_{u=j}^{j'} W_u t_{mu}.$$

Case 2: $i < m$. We compute $f_{j''}(j, j', i)$ for all $j \leq j' \leq j''$ in two separate subcases in order to get

$$f^*(j, j', i) = \min_{j \leq j'' \leq j'} \{f_{j''}(j, j', i)\}.$$

Case 2a: $i < m$ and $j'' = j'$. Jobs j, ..., j' are processed in a single setup on machine i, a setup is performed on machine $(i + 1)$ and the jobs j, ..., j' are scheduled on machines $i + 1$, ..., m. Hence,

$$f_{j''}(j, j', i) = (\tau_{jj'i} + s_{i+1})W_j + f^*(j, j', i + 1).$$

Case 2b: $i < m$ and $j'' < j'$. The processing of jobs j, ..., j' on machines i, ..., m can be viewed as a five-step process: (i) jobs j, ..., j'' are processed in a single setup on machine i; (ii) processing is terminated on machine i and machine $i+1$ is set up; (iii) jobs j, ..., j'' are processed on machines $i+1$, ..., m; (iv) processing is terminated on machine m and machine i is set up again; and (v) jobs $j''+1$, ..., j' on machines i, ..., m are scheduled. For each of the above 5 steps, we can compute a contribution to $f_{j''}(j, j', i)$ such that $f_{j''}(j, j', i)$ is obtained by summing up contribution of all the steps. The contribution of step (i) is $\tau_{jj''i}W_j$. The contribution of step (ii) is $s_{i+1}W_j$. If step (iii) is performed using a partial schedule $\sigma_{j,j'',i+1}$, the contribution of step (iii) is $f(\sigma_{j,j'',i+1})$. Hence, step (iii) can be optimally performed using the partial schedule that yields $f^*(j, j'', i+1)$. Contribution of step (iv) is $s_i W_{j''+1}$. Step (v) can be optimally performed using the partial schedule that yields $f^*(j''+1, j', i)$. Hence,

$$f_{j''}(j, j', i) = (\tau_{jj''i} + s_{i+1})W_j + f^*(j, j'', i+1) + s_i W_{j''+1} + f^*(j''+1, j', i).$$

Computation of $h(j, i)$

Now, we shall discuss the computation of $h(j, i)$.

Case 1: $i = m$. An optimal partial schedule is obtained by processing jobs j, ..., n on machine m without any setup. Hence,

$$h(j, m) = \sum_{u=j}^{n} w_u \sum_{v=j}^{u} t_{mv} = \sum_{u=j}^{n} W_u t_{mu}.$$

Case 2: $i < m$. We compute $h_{j'}(j, i)$ for all $j \leq j' \leq n$ in two separate subcases

in order to get

$$h(j,i) = \min_{j \le j' \le n} \{h_{j'}(j,i)\}.$$

Case 2a: $i < m$ and $j' = n$. Jobs j, ..., n are processed in a single setup on machine i, a setup is performed on machine $(i+1)$ and the jobs j, ..., n are scheduled on machines $i+1$, ..., m. Hence,

$$h_n(j,i) = (\tau_{jni} + s_{i+1})W_j + h(j,i+1).$$

Case 2b: $i < m$ and $j' < n$. The processing jobs j, ..., n on machines i, ..., m can be viewed as a five-step process: (i) jobs j, ..., j' are processed in a single setup on machine i; (ii) processing is terminated on machine i and machine $i+1$ is set up; (iii) jobs j, ..., j' are processed on machines $i+1$, ..., m; (iv) processing is terminated on machine m and machine i is set up again; and (v) jobs $j'+1$, ..., n on machines i, ..., m are scheduled. For each of the above 5 steps, we can compute a contribution to $h_{j'}(j,i)$ such that $h_{j'}(j,i)$ is obtained by summing up contribution of all the steps. The contribution of step (i) is $\tau_{jj'i}W_j$. The contribution of step (ii) is $s_{i+1}W_j$. If step (iii) is performed using a partial schedule $\sigma_{j,j',i+1}$, contribution of step (iii) is $f(\sigma_{j,j',i+1})$. Hence, step (iii) can be optimally performed using the partial schedule that yields $f^*(j,j',i+1)$. Contribution of step (iv) is $s_iW_{j'+1}$. Step (v) can be optimally performed using the partial schedule that yields $h(j'+1,i)$. Hence,

$$h_{j'}(j,i) = (\tau_{jj'i} + s_{i+1})W_j + f^*(j,j',i+1) + s_iW_{j'+1} + h(j'+1,i).$$

Input: $t_{ij}, w_j, s_i, \forall\ 1 \leq i \leq m$ and $1 \leq j \leq n$
Output: An optimal schedule
Compute $W_j\ \forall j$ and $\tau_{jj'i}\ \forall 1 \leq j \leq j' \leq n$ and $1 \leq i \leq m$
Compute $h(j,m) = \sum_{u=j}^{n} W_u t_{mu}\ \forall 1 \leq j \leq n$
Compute $f^*(j,j',m) = \sum_{u=j}^{j'} W_u t_{mu}\ \forall 1 \leq j \leq j' \leq n$
For $i = m-1$ down to 1 do
  For $j = n$ down to 1 do
    Compute $h(j,i) = \min_{j \leq j' \leq n} \{h_{j'}(j,i)\}$
    If $i > 1$ then compute $f^*(j,j',i) = \min_{j \leq j'' \leq j'} \{f_{j''}(j,j',i)\}\ \forall j \leq j' \leq n$
Output $s_1 W_1 + h(1,1)$

Algorithm: Problem $1Fm|s_i$,*fixed sequence*$|\sum w_j C_j$

Running Time

We precompute W_j and $\tau_{jj'i}\ \forall 1 \leq j \leq j' \leq n$ and $\forall 1 \leq i \leq m$ in times $O(n)$ and $O(mn^2)$ respectively.

Consider the case with two machines. We compute $h(j,2)\ \forall 1 \leq j \leq n$ in time $O(n)$, $f^*(j,j',2)\ \forall 1 \leq j \leq j' \leq n$ in time $O(n^2)$ and $h(j,1)\ \forall 1 \leq j \leq n$ in time $O(n^2)$. No $f^*(j,j',1)$ is computed. Hence, the algorithm requires time $O(n^2)$ if $m = 2$.

Consider the case with three or more machines. Each $f^*(j,j',i)$ with $1 \leq j \leq j' \leq n$ and $2 \leq i \leq m$ can be computed in time $O(n)$, and the number of $f^*(j,j',i)$ values is $O(mn^2)$. Hence, all $f^*(j,j',i)$ can be computed in time $O(mn^3)$. Each $h(j,i)$ with $1 \leq j \leq n$ and $1 \leq i \leq m$ can be computed in time $O(n)$ and the number of $h(j,i)$ values is $O(mn)$. Hence, all $h(j,i)$ can be computed in time $O(mn^2)$. Overall, the algorithm can be implemented in time $O(mn^3)$.

Theorem 7.1 *The problem* $1Fm|s_i$,fixed sequence$|\sum w_j C_j$ *is solved in time* $O(n^2)$

if $m = 2$ *and* $O(mn^3)$ *if* $m \geq 3$.

Corollary 7.1.1 *The problem* $1Fm|s_i, \text{aptw}|\sum w_jC_j$ *is solved in time* $O(n^2)$ *if* $m = 2$ *and* $O(mn^3)$ *if* $m \geq 3$.

However, the time complexity for the case with $m = 2$ can further be improved. Both $1F2|s_i$,*fixed sequence*$|\sum w_jC_j$ and $1O2|s_i$,*fixed sequence*$|\sum w_jC_j$ can be solved in $O(n)$ time.

7.2.2 Two-Machine Flow Shop Case

In this section we shall consider the problem $1F2|s_i$, *fixed sequence*$|\sum w_jC_j$. Recall that for the case of two machines there exists an optimal schedule which is a batching schedule. Also recall that we assume that the fixed sequence is 1, 2, ..., n and define $W_j = \sum_{u=j}^{n} w_u$.

In every schedule each operation $(1, j)$ precedes the completion of jobs $j, ..., n$ and possibly some other jobs. Hence, the contribution $\eta(1, j)$ of operation $(1, j)$ satisfies $\eta(1, j) \geq t_{1j}W_j$. Operation $(2, j)$ is the last operation of job j and precedes the completion of jobs $(j + 1), ..., n$. Hence, the contribution $\eta(2, j)$ of operation $(2, j)$ is $\eta(2, j) = t_{2j}W_j$. Let $\eta^0(1, j) = t_{1j}W_j$ and $\eta^0(2, j) = t_{2j}W_j$. We say that *contribution* η^0 *of the job-order* is $\eta^0 = \sum \eta^0(1, j) + \sum \eta^0(2, j) = \sum(t_{1j} + t_{2j})W_j$.

Suppose that a schedule has k batches. Each machine i has to be set up once before each of the k batches starts processing on machine i. Hence, we say that each

batch requires one setup on machine 1 and another on machine 2.

Consider any batch $[j, j']$. Each of the two setups of the batch, and all operations $(1, j'')$, $j \leq j'' \leq j'$, precede completion of jobs $j, ..., n$. Furthermore, each operation $(2, j'')$, $j \leq j'' \leq j'$, is the last operation of job j'' and precedes completion of jobs j'' $+1$, ..., n. We say that *contribution of batch* $[j, j']$ is the total contribution of setups and operations of the batch not included in the contribution of the job-order η^0. Hence, contribution of batch $[j, j']$ is

$$\begin{aligned}
\eta[j, j'] &= (s_1 + s_2 + \sum_{j''=j}^{j'} t_{1j''})W_j + \sum_{j''=j}^{j'} t_{2j''}W_{j''} \\
&\quad - \sum_{j''=j}^{j'} \eta^0(1, j'') - \sum_{j''=j}^{j'} \eta^0(2, j'') \\
&= (s_1 + s_2)W_j + W_j \sum_{j''=j}^{j'} t_{1j''} + \sum_{j''=j}^{j'} t_{2j''}W_{j''} \\
&\quad - \sum_{j''=j}^{j'} t_{1j''}W_{j''} - \sum_{j''=j}^{j'} t_{2j''}W_{j''} \\
&= (s_1 + s_2)W_j + \sum_{j''=j}^{j'} t_{1j''}(W_j - W_{j''}). \qquad (7.2)
\end{aligned}$$

Consider any batching schedule with batching policy $\mu = (p_1, ..., p_k)$. Let $p_0 = 0$. It follows from Remark 7.1 that the total weighted completion time is

$$\eta^0 + \sum_{1 \leq u \leq k} \eta[p_{u-1} + 1, p_u].$$

For $1 \leq j \leq n$, let $g(j)$ = minimum total contribution of the batches when jobs j,

..., n are scheduled. For a dummy job $(n+1)$ set $g(n+1) = 0$. We get $\forall\ 1 \leq j \leq n$,

$$g(j) = \min_{j \leq j' \leq n} \{\eta[j, j'] + g(j'+1)\}. \tag{7.3}$$

The minimum total weighted completion time is $g^* = \eta^0 + g(1)$. Observe that the above equation can be used recursively to compute $g(j)\ \forall\ 1 \leq j \leq (n-1)$ from $g(j+1)$, ..., $g(n)$. A straightforward implementation of such a recursion requires $O(n^2)$ time. However, as we show in the following, the array $\{\eta[j, j']\}$ satisfies Monge properties. Moreover, for any $1 \leq j \leq j' < n$, $\eta[j, j'+1]$ can be computed from $\eta[j, j']$ in constant time if all the W_j values are precomputed.

First, we shall show that the array $\{\eta[j, j']\}$ satisfies Monge property. For any $1 \leq j < j' < n$, we get

$$\begin{aligned}
&\eta[j, j'+1] + \eta[j+1, j'] - \eta[j, j'] - \eta[j+1, j'+1] \\
&\quad = (s_1 + s_2)W_j + \textstyle\sum_{j''=j}^{j'+1} t_{1j''}(W_j - W_{j''}) \\
&\qquad +(s_1 + s_2)W_{j+1} + \textstyle\sum_{j''=j+1}^{j'} t_{1j''}(W_{j+1} - W_{j''}) \\
&\qquad -(s_1 + s_2)W_j - \textstyle\sum_{j''=j}^{j'} t_{1j''}(W_j - W_{j''}) \\
&\qquad -(s_1 + s_2)W_{j+1} - \textstyle\sum_{j''=j+1}^{j'+1} t_{1j''}(W_{j+1} - W_{j''}) \\
&\quad = \textstyle\sum_{j''=j}^{j'+1} t_{1j''}(W_j - W_{j''}) + \sum_{j''=j+1}^{j'} t_{1j''}(W_{j+1} - W_{j''}) \\
&\qquad - \textstyle\sum_{j''=j}^{j'} t_{1j''}(W_j - W_{j''}) - \sum_{j''=j+1}^{j'+1} t_{1j''}(W_{j+1} - W_{j''}) \\
&\quad = t_{1(j'+1)}(W_j - W_{j'+1}) - t_{1(j'+1)}(W_{j+1} - W_{j'+1}) \\
&\quad = t_{1(j'+1)}W_j - t_{1(j'+1)}W_{j+1} \\
&\quad = t_{1(j'+1)}w_j \\
&\quad \geq 0.
\end{aligned}$$

Next, we shall show that for any $1 \leq j \leq j' < n$, $\eta[j, j'+1]$ can be computed from $\eta[j, j']$ in constant time if all the W_j values are precomputed. For any $1 \leq j \leq j' < n$,we get

$$\begin{aligned}\eta[j,j'+1]-\eta[j,j'] &= (s_1+s_2)W_j+\textstyle\sum_{j''=j}^{j'+1} t_{1j''}(W_j-W_{j''})\\ &\quad -(s_1+s_2)W_j-\textstyle\sum_{j''=j}^{j'} t_{1j''}(W_j-W_{j''})\\ &= \textstyle\sum_{j''=j}^{j'+1} t_{1j''}(W_j-W_{j''})-\sum_{j''=j}^{j'} t_{1j''}(W_j-W_{j''})\\ &= t_{1(j'+1)}(W_j-W_{j'+1}).\end{aligned}$$

Hence, the algorithms given by Galil and Park [40], Eppstein [37], or Wilber [122] can be used to compute g^* in $O(n)$ time using Equation 7.3.

Theorem 7.2 *The problem* $1F2|s_i$, fixed sequence$|\sum w_jC_j$ *is solved in* $O(n)$ *time.*

7.2.3 Two-Machine Open Shop Case

In this section we shall consider the problem $1O2|s_i$, *fixed sequence*$|\sum w_jC_j$. Recall that for the case of two machines there exists an optimal schedule which is a batching schedule . Also recall that we assume that the fixed sequence is 1, 2, ..., n and define $W_j=\sum_{u=j}^{n} w_u$. As in the case of flow shop, we define $\eta^0(1,j)=t_{1j}W_j$, $\eta^0(2,j)=t_{2j}W_j$ and *the contribution* η^0 *of the job-order* to be $\eta^0=\sum\eta^0(1,j)+\sum\eta^0(2,j)$ $=\sum(t_{1j}+t_{2j})W_j$.

Consider a schedule in which the first operation is processed on machine i_1. Let i_2 be the other machine. Initially, machine i_1 is set up. Each odd batch is started on machine i_1, and before the batch starts operation on machine i_2, machine i_2 has to be set up. Similarly, each even batch is started on machine i_2, and before the batch starts operation on machine i_1, machine i_1 has to be set up. Hence, we say that each batch starting on machine i_1 requires a setup on machine i_2 and each batch starting on machine i_2 requires a setup on machine i_1. Observe that a schedule with k batches

requires $k+1$ setups, one in the beginning and one each for the other batches.

Consider any batch $[j,j']$ that starts on machine i_1 and terminates on i_2. The setup on machine i_2 and all operations (i_1,j''), $j \le j'' \le j'$, precede the completion of jobs $j,...,n$. Furthermore, each operation (i_2,j''), $j \le j'' \le j'$, is the last operation of job j'' and precedes the completion of jobs $j''+1,...,n$. We say that *the contribution of batch* $[j,j']$ is the total contribution of setups and operations of the batch not included in the contribution of the job-order η^0. Hence, the contribution of batch $[j,j']$ starting on machine i_1 and terminating on machine i_2 is

$$\begin{aligned}
\eta_{i_1}[j,j'] &= (s_{i_2} + \sum_{j''=j}^{j'} t_{i_1 j''})W_j + \sum_{j''=j}^{j'} t_{i_2 j''} W_{j''} \\
&\quad - \sum_{j''=j}^{j'} \eta^0(1,j'') - \sum_{j''=j}^{j'} \eta^0(2,j'') \\
&= (s_{i_2} + \sum_{j''=j}^{j'} t_{i_1 j''})W_j + \sum_{j''=j}^{j'} t_{i_2 j''} W_{j''} \\
&\quad - \sum_{j''=j}^{j'} t_{i_1 j''} W_{j''} - \sum_{j''=j}^{j'} t_{i_2 j''} W_{j''} \\
&= s_{i_2} W_j + \sum_{j''=j}^{j'} t_{i_1 j''}(W_j - W_{j''}). \qquad (7.4)
\end{aligned}$$

Consider any batching schedule with batching policy $\mu = (p_1,...,p_k)$ and starting on machine i_1. Let i_2 be the other machine. Let $p_0 = 0$. It follows from Remark 7.1 that the total weighted completion time is

$$\eta^0 + s_{i_1} W_1 + \sum_{u=1,3,...} \eta_{i_1}[p_{u-1}+1, p_u] + \sum_{u=2,4,...} \eta_{i_2}[p_{u-1}+1, p_u].$$

For $1 \leq j \leq n$, let $g(j, i^*)$ = minimum total contribution of the batches when jobs $j, ..., n$ are scheduled and post-setup processing of operation (i^*, j) starts at time zero. For a dummy job $(n+1)$ set $g(n+1, 1) = g(n+1, 2) = 0$. We get $\forall\ 1 \leq j \leq n$,

$$g(j,1) = \min_{j \leq j' \leq n} \{\eta_1[j,j'] + g(j'+1, 2)\} \tag{7.5}$$

and

$$g(j,2) = \min_{j \leq j' \leq n} \{\eta_2[j,j'] + g(j'+1, 1)\}. \tag{7.6}$$

The minimum total weighted completion time is $g^* = \eta^0 + \min\{s_1 + g(1,1),\ s_2 + g(1,2)\}$. Observe that the above equations can be recursively used to compute $g(j,1)$ $\forall\ 1 \leq j \leq n$ from $g(j+1,2)$, ..., $g(n+1,2)$ and $g(j,2)$ $\forall\ 1 \leq j \leq n$ from $g(j+1,1)$, ..., $g(n+1,1)$. A straightforward implementation of such a recursion requires $O(n^2)$ time. However, as in the flow shop case, we can show that each array $\{\eta_1[j,j']\}$ and $\{\eta_2[j,j']\}$ satisfies the Monge property. Moreover, for any $1 \leq j \leq j' < n, \eta_1[j, j'+1]$ and $\eta_2[j, j'+1]$ can be obtained from $\eta_1[j,j']$ and $\eta_2[j,j']$ in constant time if all the values W_j are precomputed.

First, we shall show that each array $\{\eta_1[j,j']\}$ and $\{\eta_2[j,j']\}$ satisfies the Monge property. For any $1 \leq j < j' < n$, we get

$\eta_1[j, j'+1] + \eta_1[j+1, j'] - \eta_1[j,j'] - \eta_1[j+1, j'+1]$

$$
\begin{aligned}
&= s_2 W_j + \sum_{j''=j}^{j'+1} t_{1j''}(W_j - W_{j''}) + s_2 W_{j+1} + \sum_{j''=j+1}^{j'} t_{1j''}(W_{j+1} - W_{j''}) \\
&\quad -s_2 W_j - \sum_{j''=j}^{j'} t_{1j''}(W_j - W_{j''}) - s_2 W_{j+1} - \sum_{j''=j+1}^{j'+1} t_{1j''}(W_{j+1} - W_{j''}) \\
&= \sum_{j''=j}^{j'+1} t_{1j''}(W_j - W_{j''}) + \sum_{j''=j+1}^{j'} t_{1j''}(W_{j+1} - W_{j''}) \\
&\quad - \sum_{j''=j}^{j'} t_{1j''}(W_j - W_{j''}) - \sum_{j''=j+1}^{j'+1} t_{1j''}(W_{j+1} - W_{j''}) \\
&= t_{1(j'+1)}(W_j - W_{j'+1}) - t_{1(j'+1)}(W_{j+1} - W_{j'+1}) \\
&= t_{1(j'+1)}(W_j - W_{j+1}) \\
&= t_{1(j'+1)} w_j \\
&\geq 0
\end{aligned}
$$

Similarly, for any $1 \leq j < j' < n$, we get $\eta_2[j, j'+1] + \eta_2[j+1, j'] - \eta_2[j, j'] - \eta_2[j+1, j'+1] \geq 0$.

Next we shall show that for any $1 \leq j \leq j' < n, \eta_1[j, j'+1]$ and $\eta_2[j, j'+1]$ can be obtained from $\eta_1[j, j']$ and $\eta_2[j, j']$ in constant time if all the values W_j are precomputed. For any $1 \leq j \leq j' < n$, we get

$$
\begin{aligned}
\eta_1[j, j'+1] - \eta_1[j, j'] &= s_2 W_j + \sum_{j''=j}^{j'+1} t_{1j''}(W_j - W_{j''}) \\
&\quad -s_2 W_j - \sum_{j''=j}^{j'} t_{1j''}(W_j - W_{j''}) \\
&= \sum_{j''=j}^{j'+1} t_{1j''}(W_j - W_{j''}) - \sum_{j''=j}^{j'} t_{1j''}(W_j - W_{j''}) \\
&= t_{1(j'+1)}(W_j - W_{j'+1})
\end{aligned}
$$

Hence, $\eta_1[j, j'+1]$ can be obtained from $\eta_1[j, j']$ in constant time if all the values W_j are precomputed. Similarly, we can show that $\eta_2[j, j'+1]$ can be obtained from $\eta_2[j, j']$ in constant time if all the values W_j are precomputed.

Thus, the algorithm given by Eppstein [37] can be used to compute g^* in $O(n)$ time using Equations 7.5 and 7.6.

Theorem 7.3 *The problem* $1O2|s_i$, fixed sequence$|\sum w_j C_j$ *is solved in* $O(n)$ *time.*

7.2.4 Network Representation for the Two-Machine Cases

Fixed-sequence cases with objectives $L_{\max}$ and $\sum w_jC_j$ can be given a similar network representation. In fact, the network representation is more intuitive in the case of objective $\sum w_jC_j$. For example, in the case of objective $\sum w_jC_j$, each arc is associated with a single weight and the length of the path is obtained by summing up the weights of all the arcs in the path. Consequently, we get a classical shortest path problem.

The graphs defined in Chapter 5 for problems $1O2|s_i|L_{\max}$ and $1F2|s_i|L_{\max}$ apply to problems $1O2|s_i$, *fixed sequence*$|\sum w_jC_j$ and $1F2|s_i$, *fixed sequence*$|\sum w_jC_j$ respectively. For ease of reading, we restate the definitions of the graphs here.

Consider the open shop case first. Introduce a dummy job $(n+1)$. Define a directed network $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with node-set $\mathbf{V} = \{0\} \cup \mathbf{V}_1 \cup \mathbf{V}_2$ and arc-set $\mathbf{E} = \mathbf{E}_0 \cup \mathbf{E}_1 \cup \mathbf{E}_2$. For each operation $(1, j)$, there is a node a_j in $\mathbf{V}_1$. For each operation $(2, j)$, there is a node b_j in $\mathbf{V}_2$. Arc-set $\mathbf{E}_0$ contains only two arcs $\langle 0, a_1\rangle$ and $\langle 0, b_1\rangle$. For all $1 \leq j < j' \leq (n+1)$ there is an arc $\langle a_j, b_{j'}\rangle$ in $\mathbf{E}_1$ and an arc $\langle b_j, a_{j'}\rangle$ in $\mathbf{E}_2$.

Each arc e is associated with a single weight $r(e)$ which represents a contribution to $\sum w_jC_j$. If $e = \langle a_j, b_{j'}\rangle$, e represents processing of batch $[j, j'-1]$ starting on machine 1. Hence, $r(e) = \eta_1[j, j'-1]$. If $e = \langle b_j, a_{j'}\rangle$, e represents processing of batch $[j, j'-1]$ starting on machine 2. Hence, $r(e) = \eta_2[j, j'-1]$. If $e = \langle 0, a_1\rangle$, e represents the initial setup on machine 1. Hence, $r(e) = s_1W_1$. If $e = \langle 0, b_1\rangle$, e represents the initial setup on machine 2. Hence, $r(e) = s_2W_1$. The problem is to

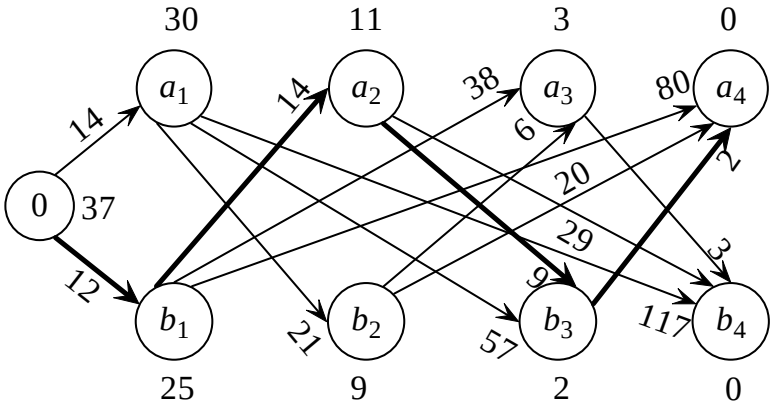

Figure 7.1: A network representation of the two-machine open shop problem

compute a shortest path from node 0 to node a_{n+1} or b_{n+1}. The quantities $g(j,1)$ and $g(j,2)$ give the lengths of the shortest paths from nodes a_j and b_j, respectively.

For example, consider the data shown in Example 7.1 below.

Example 7.1 *Consider the following problem involving* $n = 3$ *jobs with* $s_1 = 2$ *and* $s_2 = 3$.

j	1	2	3
t_{1j}	8	9	10
t_{2j}	2	6	7
w_j	4	2	1

There are three jobs 1, 2 and 3. Introduce a dummy job 4. The nodes are 0, a_1, ..., a_4 and b_1, ..., b_4. The corresponding graph and arc weights are shown in Figure 7.1. First, compute $W_3 = w_3 = 1$, $W_2 = w_2 + w_3 = 1 + 2 = 3$ and $W_1 = w_1 + w_2 + w_3 = 1 + 2 + 4 = 7$. Arc weights are computable functions of t_{ij}, s_i, and W_j. For example,

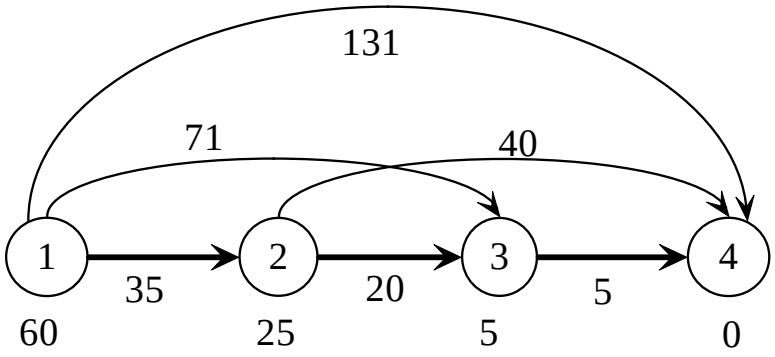

Figure 7.2: A network representation of the two machine flow shop problem

$$\begin{aligned} r(a_1, b_3) &= \eta_1[1, 3-1] = \eta_1[1, 2] \\ &= s_2 W_1 + t_{11}(W_1 - W_1) + t_{12}(W_1 - W_2) \quad \text{from Equation 7.4} \\ &= 3(7) + 8(7-7) + 9(7-3) = 57 \end{aligned}$$

The shortest paths are computed first from nodes a_3 and b_3, then from nodes a_2 and b_2 and so on. For example, the length of a shortest path from node a_1,

$$\begin{aligned} g(1,1) &= \min\{r(a_1, b_2) + g(2,2), r(a_1, b_3) + g(3,2), r(a_1, b_4) + g(4,2)\} \\ &= \min\{21 + 9, 57 + 2, 117 + 0\} = 30 \end{aligned}$$

A shortest path from node 0 is $\langle \langle 0, b_1 \rangle, \langle b_1, a_2 \rangle, \langle a_2, b_3 \rangle, \langle b_3, a_4 \rangle \rangle$. Arc $\langle 0, b_1 \rangle$ represents setting up of machine 2, arc $\langle b_1, a_2 \rangle$ represents batch $[1, 1]$, arc $\langle a_2, b_3 \rangle$ represents batch $[2, 2]$ and arc $\langle b_3, a_4 \rangle$ represents batch $[3, 3]$. Hence, an optimal solution is to start from machine 2 and use batching policy $(1, 2, 3)$. The corresponding sequence of operations is $(2, 1)$, $(1, 1)$, $(1, 2)$, $(2, 2)$, $(2, 3)$, $(1, 3)$.

Now, consider the flow shop case. Again, introduce a dummy job $(n+1)$. Define a directed network $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ is the node-set and $\mathbf{E}$ is the arc-set. For each job j, there is a node j. For each pair of nodes j and j' with $1 \leq j < j' \leq (n+1)$,

there is an arc $\langle j, j' \rangle$ which represents processing of batch $[j, j'-1]$ and is associated with weight $r(j, j') = \eta[j, j'-1]$. The problem is to compute a shortest path from node 1 to $(n+1)$. The quantity $g(j)$ gives the length of the shortest path from node j.

Considering the data shown in Example 7.1, we illustrate the network representation in Figure 7.2. There are three jobs 1, 2 and 3. Introduce a dummy job 4. The nodes are 1, 2, 3, and 4. As in the open shop case, $W_3 = 1$, $W_2 = 3$, $W_1 = 7$ and arc weights are computable functions of t_{ij}, s_i, and W_j. For example,

$$\begin{aligned} r(1,3) &= \eta[1, 3-1] = \eta[1,2] \\ &= (s_1 + s_2)W_1 + t_{11}(W_1 - W_1) + t_{12}(W_1 - W_2) \quad \text{from Equation 7.2} \\ &= (2+3)(7) + 8(7-7) + 9(7-3) = 71 \end{aligned}$$

The shortest paths are computed first from node 3, then from node 2 and so on. For example, the length of a shortest path from node 1,

$$\begin{aligned} g(1) &= \min\{r(1,2) + g(2), r(1,3) + g(3), r(1,4) + g(4)\} \\ &= \min\{35 + 25, 71 + 5, 131 + 0\} = 60 \end{aligned}$$

A shortest path from node 1 is $\langle \langle 1,2 \rangle, \langle 2,3 \rangle, \langle 3,4 \rangle \rangle$. Arc $\langle 1,2 \rangle$ represents batch $[1,1]$, arc $\langle 2,3 \rangle$ represents batch $[2,2]$ and arc $\langle 3,4 \rangle$ represents batch $[3,3]$. Hence, an optimal solution is to use batching policy $(1,2,3)$. The corresponding sequence of operations is $(1,1)$, $(2,1)$, $(1,2)$, $(2,2)$, $(1,3)$, $(2,3)$.

The algorithms for problems $1O2|s_i|L_{\max}$ and $1F2|s_i|L_{\max}$ presented in Chapter 5 are based on two crucial rules: the *node elimination* rule and the *arc elimination* rule. The fact that the node elimination rule still applies to the cases with objective $\sum w_j C_j$ follows from the Monge property. However, the arc elimination rule does not

apply to the cases with objective $\sum w_j C_j$. We shall show this in the context of a flow shop. The open shop case is similar.

First, consider the node elimination rule. Recall that $g(j)$ is the minimum total contribution of the batches when jobs j, ..., n are scheduled. As we have shown that the Monge property applies, for any $1 \leq j < j' < n$, we get $\eta[j, j' +1] + \eta[j +1, j']$ $-\eta[j, j'] - \eta[j+1, j'+1] \geq 0$. By rearranging terms,

$$\begin{array}{lcl} \eta[j, j'+1] & \geq & \eta[j, j'] + \eta[j+1, j'+1] - \eta[j+1, j'], \text{ or} \\ r(j, j'+2) & \geq & r(j, j'+1) + r(j+1, j'+2) - r(j+1, j'+1), \text{ or} \\ r(j, j'+2) + g(j'+2) & \geq & r(j, j'+1) + g(j'+1) \\ & & +r(j+1, j'+2) + g(j'+2) \\ & & -r(j+1, j'+1) - g(j'+2) \end{array}$$

Hence, $r(j +1, j'+2) + g(j' +2) \geq r(j +1, j' +1) + g(j' +1) \Rightarrow r(j, j' +2) + g(j' +2)$ $\geq r(j, j' +1) + g(j' +1)$. Thus, if we have $r(j +1, j' +2) + g(j' +2) \geq r(j +1, j' +1)$ $+g(j' +1)$, then node $(j' +2)$ can be eliminated from computation of a shortest path from nodes 1, ..., j.

Now, consider the arc elimination rule. Consider the following example:

Example 7.2 *Consider an instance of the problem* $1F2|s_i,$ fixed sequence$|\sum w_j C_j$ *with* $s_1 = s_2 = 2$, $w_j = 1$ *and the following processing times:*

j	1	2	3
t_{1j}	1	10	1
t_{2j}	1	1	20

For various batching policies , the corresponding path and completion times are summarized below:

Batching policy, μ	Corresponding path	C_1	C_2	C_3	$\sum w_j C_j$
$(1,3)$	$\langle 1,2,4\rangle$	6	22	42	70
$(2,3)$	$\langle 1,3,4\rangle$	16	17	42	75
(3)	$\langle 1,4\rangle$	17	18	38	73

Consider the computation of a shortest path from node 1. A choice of arc $\langle 1,4\rangle$ gives a solution with $\sum w_j C_j = 73$. Arc $\langle 1,4\rangle$ is a better choice than arc $\langle 1,3\rangle$ which gives a solution with $\sum w_j C_j = 75$. However, arc $\langle 1,2\rangle$ gives a solution with $\sum w_j C_j = 70$ which turns out to be the unique optimum. This shows that the arc elimination rule does not apply to cases with objective $\sum w_j C_j$.

7.3 The Fixed Batching Policy Case

In this section we shall consider the cases with known and fixed batching policy. We shall show that problems $1F2|s_i$,*fixed batching policy*$|\sum w_j C_j$ and $1F2|s_i$,*fixed batching policy*$|\sum w_j C_j$ are $\mathcal{NP}$-hard. However, both the flow shop and open shop cases are solvable if $w_j = 1$.

7.3.1 Weighted Completion Time

Theorem 7.4 *Both* $1F2|s_i$,fixed batching policy$|\sum w_j C_j$ *and* $1F2|s_i$,fixed batching policy$|\sum w_j C_j$ *are* $\mathcal{NP}$*-hard.*

Proof: Given an instance of the partition problem with set of integers $\mathbf{A} = \{a_1, ..., a_k\}$, $\sum_{a_l \in \mathbf{A}} a_l = 2b$ we define an instance each of $1F2|s_i$,*fixed batching policy*$|\sum w_j C_j$ and $1F2|s_i$,*fixed batching policy*$|\sum w_j C_j$ as follows:

$$
\begin{array}{lcll}
n & = & 2k-2 & \\
\mu & = & (k-1, 2k-2) & \\
S_1 = S_2 & = & 0 & \\
t_{1j} = t_{2j} = w_j & = & \left\{ \begin{array}{ll} a_j & \text{for } j = 1, 2, ...k \\ 0 & \text{for } j = (k+1), (k+2), ..., (2k-2) \end{array} \right. &
\end{array}
$$

Consider any schedule σ with job-order ψ. Let $b_j = t_{1j} = t_{2j} = w_j$. Let $\mathbf{J}_1$ and $\mathbf{J}_2$ be the set of jobs in batches 1 and 2 respectively. Let $x = \sum_{p=1}^{k-1} b_{\psi(p)}$. For any position $p \leq (k-1)$, completion time $C_{\psi(p)} = x + \sum_{p'=1}^{p} b_{\psi(p')}$. Again, for any position $p > (k-1)$, completion time $C_{\psi(p)} = 2b + \sum_{p'=1}^{p} b_{\psi(p')}$. Hence, the total weighted completion time is

$$
\begin{array}{rcl}
\sum w_j C_j & = & \sum_{p=1}^{k-1} b_{\psi(p)} (x + \sum_{p'=1}^{p} b_{\psi(p')}) \\
 & & + \sum_{p=k}^{2k-2} b_{\psi(p)} (2b + \sum_{p'=1}^{p} b_{\psi(p')}) \\
 & = & \sum_{p=1}^{k-1} b_{\psi(p)} x + \sum_{p=1}^{k-1} b_{\psi(p)} \sum_{p'=1}^{p} b_{\psi(p')} \\
 & & + 2b \sum_{p=k}^{2k-2} b_{\psi(p)} + \sum_{p=k}^{2k-2} b_{\psi(p)} \sum_{p'=1}^{p} b_{\psi(p')} \\
 & = & x^2 + \sum_{p=1}^{k-1} b_{\psi(p)} \sum_{p'=1}^{p} b_{\psi(p')} \\
 & & + 2b \sum_{p=k}^{2k-2} b_{\psi(p)} + \sum_{p=k}^{2k-2} b_{\psi(p)} \sum_{p'=1}^{p} b_{\psi(p')} \\
 & = & x^2 + \sum_{p=1}^{k-1} b_{\psi(p)} \sum_{p'=1}^{p} b_{\psi(p')} + 2b(2b - x) + \sum_{p=k}^{2k-2} b_{\psi(p)} \sum_{p'=1}^{p} b_{\psi(p')} \\
 & = & x^2 + 2b(2b - x) + \sum_{p=1}^{2k-2} b_{\psi(p)} \sum_{p'=1}^{p} b_{\psi(p')} \\
 & = & x^2 - 2bx + 4b^2 + \sum_{j=1}^{2k-2} b_j \sum_{j'=1}^{j} b_{j'} \\
 & & x^2 - 2bx + 4b^2 + \sum_{j=1}^{k} a_j \sum_{j'=1}^{j} a_{j'}
\end{array}
$$

Setting $d \sum w_j C_j / dx = 0$, we get that $\sum w_j C_j$ attains a minimum value of $3b^2 + \sum_{j=1}^{k} a_j \sum_{j'=1}^{j} a_{j'}$ if and only if $x = b$. Thus, the problem of checking whether there exists a job-order with $\sum w_j C_j \leq 3b^2 + \sum_{j=1}^{k} a_j \sum_{j'=1}^{j} a_{j'}$ is equivalent to the partition problem. ■

7.3.2 Flow Shop with Total Completion Time

Consider the problem $1F2|s_i, \textit{fixed batching policy}| \sum C_j$. Suppose that the given batching policy is $\mu = (p_1, ..., p_k)$. Let $p_0 = 0$. For any p with $p_{u-1} < p \leq p_u$, if

any job j is assigned to position p, then job j belongs to the u-th batch. Consider any job-order ψ. Each machine i is set up before each operation $(i, \psi(p_{u-1}+1))$, where $1 \leq u \leq k$. Since such a setup precedes completion of $(n - p_{u-1})$ jobs $\psi(p_{u-1}+1)$, ..., $\psi(n)$, the contribution of the setup is $s_i(n - p_{u-1})$. Hence the contribution of all setups is $\sum_{u=1}^{k}(s_1 + s_2)(n - p_{u-1})$.

Since the contribution of setups is a constant, the total weighted completion time is minimized by minimizing the total contribution of operations. Suppose that job j is assigned to position p belonging to the u-th batch. We get $p_{u-1} < p \leq p_u$. Operation $(1, j)$ is processed before the completion of $(n - p_{u-1})$ jobs $\psi(p_{u-1}+1)$, ..., $\psi(n)$. Operation $(2, j)$ is the last operation of job j and precedes completion of the other $(n - p)$ jobs $\psi(p+1)$, ..., $\psi(n)$. Hence, the contribution of assigning job j to position p is $\eta_{jp} = t_{1j}(n - p_{u-1}) + t_{2j}(n - p + 1)$. Hence, we get the following:

Theorem 7.5 *The problem* $1F2|s_i$,fixed batching policy$|\sum C_j$ *reduces to an assignment problem.*

For example, consider the data shown in Example 7.3 below.

Example 7.3 *Consider the following problem involving* $n = 3$ *jobs with* $s_1 = 2$ *and* $s_2 = 3$.

j	1	2	3
t_{1j}	8	9	7
t_{2j}	2	6	9
w_j	1	1	1

Suppose that the given batching policy is $(2,3)$. Since the batching policy is (2,3), we have $p_1 = 2$ and $p_2 = 3$. According to our above discussion, the problem of finding an optimal job-order reduces to an assignment problem with the following cost matrix:

		Position, p		
		1	2	3
Job, j	1	8(3)+2(3)=30	8(3)+2(2)=28	8(1)+2(1)=10
	2	9(3)+6(3)=45	9(3)+6(2)=39	9(1)+6(1)=15
	3	7(3)+9(3)=48	7(3)+9(2)=39	7(1)+9(1)=16

For example, consider the contribution of assigning job 1 to position 2. Since position 2 belongs to the first batch, $u = 1$, and, therefore, $p_{u-1} = p_0 = 0$. The contribution of assigning job 1 to position 2 is $\eta_{12} = t_{11}(n - p_{u-1}) + t_{21}(n - p + 1) = 8(3 - 0)$ $+2(3 - 2 + 1) = 8(3) + 2(2) = 28$.

An optimal assignment is as follows: job 1 to position 1, job 2 to position 3 and job 3 to position 2. This corresponds to job-order (1,3,2) with the contribution of operations = 30+15+39 = 84. The contribution of setups = $(s_1 + s_2)(n - p_0)$ $+(s_1+s_2)(n-p_1) = (2+3)(3-0) + (2+3)(3-2) = 20$. Job-order (1,3,2) and batching policy (2,3) gives the sequence of operations $(1,1)$, $(1,3)$, $(2,1)$, $(2,3)$, $(1,2)$, $(2,2)$ with a total completion time 84+20=104.

7.3.3 Open Shop with Total Completion Time

Consider the problem $1O2|s_i$,*fixed batching policy*$|\sum C_j$. Suppose that the given batching policy is $\mu = (p_1, ..., p_k)$. Let $p_0 = 0$. For any p with $p_{u-1} < p \leq p_u$, if any job j is assigned to position p, then job j belongs to the u-th batch. Consider

any job-order ψ.

Suppose that machine i_1 is set up in the beginning. Let i_2 be the other machine. The contribution of the first setup is ns_{i_1}. Each machine i is set up before each operation $(i, \psi(p_{u-1}+1))$, where $1 \leq u \leq k$, u is odd if $i = i_2$ and u is even if $i = i_1$. Since such a setup precedes completion of $(n - p_{u-1})$ jobs $\psi(p_{u-1}+1)$, ..., $\psi(n)$, the contribution of the setup is $s_i(n - p_{u-1})$. Hence, the contribution of all setups is ns_{i_1} $+ \sum_{u=2,4,\ldots} s_{i_1}(n - p_{u-1}) + \sum_{u=1,3,\ldots} s_{i_2}(n - p_{u-1})$.

Since the contribution of setups is a constant, the total weighted completion time is minimized by minimizing the total contribution of operations. Suppose that job j is assigned to position p belonging to the u-th batch. We get $p_{u-1} < p \leq p_u$.

If u is odd, then operation (i_1, j) precedes completion of $(n - p_{u-1})$ jobs $\psi(p_{u-1}+1)$, ..., $\psi(n)$. Operation (i_2, j) is the last operation of job j and precedes completion of the other $(n-p)$ jobs $\psi(p+1)$, ..., $\psi(n)$. Hence, the contribution of assigning job j to position p is $\eta_{jp}(i_1) = t_{i_1 j}(n - p_{u-1}) + t_{i_2 j}(n - p + 1)$.

If u is even, then operation (i_2, j) precedes completion of $(n - p_{u-1})$ jobs $\psi(p_{u-1}+1)$, ..., $\psi(n)$. Operation (i_1, j) is the last operation of job j and precedes completion of the other $(n-p)$ jobs $\psi(p+1)$, ..., $\psi(n)$. Hence, the contribution of assigning job j to position p is $\eta_{jp}(i_1) = t_{i_2 j}(n - p_{u-1}) + t_{i_1 j}(n - p + 1)$.

Thus, the problem $1O2|s_i$,*fixed batching policy*$|\sum C_j$ with the first setup on machine i_1 is solved by solving an assignment problem. Hence, we get the following:

Theorem 7.6 *The problem* $1O2|s_i$,fixed batching policy$|\sum C_j$ *is solved by solving two assignment problems.*

Considering the data shown in Example 7.3, we shall now illustrate the method. Suppose that the given batching policy is $(2,3)$. As it is in the flow shop case, $p_1 = 2$, $p_2 = 3$. The problem of finding an optimal job-order reduces to two assignment problems. One corresponds to starting on machine 1 and the other corresponds to starting on machine 2.

If the operator starts on machine 1, the problem of finding an optimal job-order reduces to an assignment problem with the cost matrix exactly same as what we obtained in the flow shop case (such equivalence will not hold for all n and all batching policies). Consequently, an optimal job-order is (1,3,2) with the contribution of operations = 30+39+15 = 84. The contribution of setups = $ns_1+s_2(n-p_0)+s_1(n-p_1)$ = $3(2) + 3(3-0) + 2(3-2) = 17$. Hence, the total completion time =84+17=101.

If the operator starts on machine 2, the problem of finding an optimal job-order reduces to an assignment problem with the following cost matrix:

		Position, p		
		1	2	3
Job, j	1	8(3)+2(3)=30	8(2)+2(3)=23	8(1)+2(1)=10
	2	9(3)+6(3)=45	9(2)+6(3)=36	9(1)+6(1)=15
	3	7(3)+9(3)=48	7(2)+9(3)=41	7(1)+9(1)=16

An optimal assignment is as follows: job 1 to position 1, job 2 to position 2 and job 3 to position 3. This corresponds to job-order (1,2,3) with the contribution of operations

$= 30+36+16 = 82$. The contribution of setups $= ns_2 + s_1(n - p_0) + s_2(n - p_1) = 3(3) + 2(3 - 0) + 3(3 - 2) = 18$. Hence, the total completion time $=82+18=100$.

Thus, starting on machine 2 is better than starting on machine 1. The operator starts on machine 2 and uses a job-order (1,2,3). Since the batching policy is $(2, 3)$, the sequence of operations is $(2, 1)$, $(2, 2)$, $(1, 1)$, $(1, 2)$, $(1, 3)$, $(2, 3)$. The resulting total completion time is 100.

7.4 Summary

In this chapter we consider the problems with the total (weighted) completion time objective. First, we revisit some fixed-sequence cases and improve the running time. Fixed-sequence cases with objectives $L_{\max}$ and $\sum w_j C_j$ can be given a similar network representation. In the case of objective $\sum w_j C_j$, each arc is associated with a single weight and the length of the path is obtained by summing up the weights of all the arcs in the path. Consequently, we get a classical shortest path problem.

An obvious implementation of the shortest path algorithm requires $O(n^2)$ time. We ask whether the $O(n)$ time dynamic programming approach developed in Chapter 5 can be extended. The answer is negative because the arc elimination rule does not apply although the node elimination rule applies and it is possible to compute arc weights from the previously computed arc weights in constant time. Still, it is possible to solve the resulting shortest path problem in $O(n)$ time. Because the cost matrices

satisfy what is called the Monge properties.

One motivation for analyzing the fixed-sequence cases is that it is sometimes possible to obtain a job-order which dominates all the other job-orders. For example, it follows from Theorem 4.2 that one such case is that of *agreeable processing time and weight*. Another motivation is to eventually obtain an enumeration scheme. However, as there are $n!$ job-orders, we seek an alternate scheme.

Such an alternate scheme is to enumerate over all batching policies . There are 2^{n-1} batching policies . Unfortunately, as we show in Theorem 7.4, both the flow shop and open shop problems with the objective $\sum w_j C_j$ are $\mathcal{NP}$-hard even if the batching policy is fixed. However, the cases with all $w_j = 1$ are efficiently solvable. The problem $1F2|s_i$,*fixed batching policy*$|\sum C_j$ is solved by solving an assignment problem and the problem $1O2|s_i$,*fixed batching policy*$|\sum C_j$ is solved by solving two assignment problems. Thus the cases with fixed bathing policy are solvable in $O(n^3)$ time.

In the next chapter, we develop a branch and bound algorithm for the problem $1F2|s_i|\sum C_j$ using the results on the flow shop fixed-sequence and fixed batching policy cases with the objective $\sum C_j$.

In Table 7.1, we summarize the results on various cases of the total (weighted) completion time objective.

Problem	**Result (Chapter 7)**	**Previously Known Result**
$1Fm\|s_i$,*fixed sequence*$\|\sum w_jC_j$	$O(mn^3)$	—
$1F2\|s_i$,*fixed sequence*$\|\sum w_jC_j$	$O(n)$	$O(n)$ (Albers and Brucker [4], Coffman et al. [31])
$1O2\|s_i$,*fixed sequence*$\|\sum w_jC_j$	$O(n)$	$O(n^2)$ (Gerodimos et al. [45, 46], Julien and Magazine [64], Sung and Park [111])
$1F2\|s_i$,*fixed batching policy*$\|\sum w_jC_j$	$\mathcal{NP}$-hard	—
$1O2\|s_i$,*fixed batching policy*$\|\sum w_jC_j$	$\mathcal{NP}$-hard	—
$1F2\|s_i$,*fixed batching policy*$\|\sum C_j$	$O(n^3)$	—
$1O2\|s_i$,*fixed batching policy*$\|\sum C_j$	$O(n^3)$	—

Table 7.1: Summary of results on two-machine cases with the total (weighted) completion time objective

Chapter 8

Enumeration Schemes for $1F2|s_i|\sum C_j$

In this chapter, we shall discuss a heuristic, a branch and bound scheme and an integer programming formulation. We shall discuss computational experience for various implementations of the branch and bound scheme and the integer programming formulation.

8.1 Definitions

Definition 8.1 *If* $\tilde{\mu} = (0)$ *or* $\tilde{\mu} = (p_1, ..., p_k)$ *for some* $1 \leq p_1 < ... < p_k \leq n$, *then* $\tilde{\mu}$ *is a* partial batching policy. *If* $\tilde{\mu} = (p_1, ..., p_k)$ *with* $p_k = n$, *then* $\tilde{\mu}$ *is a* complete batching policy.

The partial batching policy $\tilde{\mu} = (0)$ does not define the size of any batches. However, a partial batching policy $\tilde{\mu} = (p_1, ..., p_k)$ with $k \geq 1$ defines the size of each of the first k batches. Consider a partial batching policy $\tilde{\mu} = (p_1, ..., p_k)$ with $k \geq 1$.

Let $p_0 = 0$. For each $1 \leq u \leq k$, the size of the u-th batch is $(p_u - p_{u-1})$.

Definition 8.2 *A complete batching policy μ is* compatible *with a partial batching policy $\tilde{\mu} = (p_1, ..., p_k)$ if the size of each of the first k batches is same in both $\tilde{\mu}$ and μ. Every complete batching policy is compatible with partial batching policy $\tilde{\mu} = (0)$.*

Consider any schedule σ with job-order ψ and a complete batching policy $\mu = (p_1, ..., p_k)$. Let $p_0 = 0$.

The contribution of a batch $[p, p']$ is $\eta[p, p'] = (s_1 + s_2)W_p + \sum_{p''=p}^{p'} t_{1\psi(p'')}(W_p - W_{p''})$. Since $w_j = 1\ \forall j$, we have $W_p = (n - p + 1)$ and $W_p - W_{p''} = (n - p + 1) - (n - p'' + 1) = p'' - p$.

Hence, $\eta[p, p'] = (s_1 + s_2)(n - p + 1) + \sum_{p''=p}^{p'} t_{1\psi(p'')}(p'' - p)$.

Define

$$\eta[p, p', \psi] = \sum_{p''=p}^{p'} t_{1\psi(p'')}(p'' - p).$$

The contribution of batch $[p, p']$ is $\eta[p, p'] = (s_1 + s_2)(n - p + 1) + \eta[p, p', \psi]$.

The contribution of batching policy μ is $\eta(\mu) = \sum_{u=1}^{k}(s_1 + s_2)(n - p_{u-1}) + \sum_{u=1}^{k} \eta[p_{u-1} + 1, p_u, \psi]$.

The contribution of job-order ψ is $\eta(\psi) = \sum_{p=1}^{n}(n - p + 1)(t_{1\psi(p)} + t_{2\psi(p)})$.

The total completion time is $\Delta(\sigma) = \eta(\psi) + \eta(\mu)$.

8.2 A Heuristic

A good heuristic is important for a successful implementation of the branch and bound scheme. We use our analysis of the fixed-sequence and fixed batching policy cases to develop a heuristic. The basic idea is to arrange the jobs in the shortest total processing time order, ψ^* and obtain a batching policy μ^* which is optimal for job-order ψ^*. We then obtain a job-order which is optimal for batching policy μ^* by solving the problem $1F2|s_i$, *fixed batching policy*$|\sum C_j$ with batching policy μ^*.

We recognize the fact that jobs within a batch are processed in the ascending order of t_{2j}. Thus, if jobs j and j' are processed in the same batch and $t_{2j} < t_{2j'}$, then job j is processed before j'. Let ψ^* be a job-order such that $p < p' \Rightarrow (t_{1\psi^*(p)} + t_{2\psi^*(p)}) \leq (t_{1\psi^*(p')} + t_{2\psi^*(p')})$. In steps 1, 2 and 3 below, we compute μ^* by using a slight modification of the algorithm for $1F2|s_i$, *fixed sequence*$|\sum C_j$.

Step 1: For all $1 \leq p \leq p' \leq n$, compute $\hat{\eta}[p,p'] = (s_1 + s_2 + \sum_{p''=p}^{p'} t_{1\psi^*(p'')})(n - p + 1) + \sum_{p''=p}^{p'} t_{2\psi(p'')}(n - p'' + 1)$, where ψ is an arrangement of jobs $\psi^*(p)$, ..., $\psi^*(p')$ so that $t_{2\psi(p)} \leq ... \leq t_{2\psi(p')}$.

Step 2: Let $g(n+1) = 0$, $g(n) = \hat{\eta}[n,n]$. For $p = (n-1)$ downto 1 compute $succ(p) = \arg\min_{p \leq p' \leq n}\{\hat{\eta}[p,p'] + g(p'+1)\}$ and $g(p) = \hat{\eta}[p, succ(p)] + g(succ(p)+1)$.

Step 3: Let $k = 0$ and $p_0 = 0$. While $p_k \neq n$, set $p_{k+1} = succ(p_k + 1)$ and increase k by 1.

Step 4: Solve the problem $1F2|s_i$, *fixed batching policy*$|\sum C_j$ with batching policy

$\mu^* = (p_1, ..., p_k)$.

In step 1, quantities $\hat{\eta}[p, p']$ are computed such that for any complete batching policy $\mu = (p_1, ..., p_k)$ the total completion time is $\sum_{u=1}^{k} \hat{\eta}[p_{u-1} + 1, p_u]$ if the jobs within a batch are processed in the ascending order of t_{2j}. In step 2 a dynamic programming recursion is applied to compute the total completion time given by a batching policy μ^* which is optimal for job-order ψ^*. In step 3 the batching policy $\mu^* = (p_1, ..., p_k)$ is constructed. In step 4, the algorithm for the fixed batching policy case is applied to obtain a job-order which is optimal for the complete batching policy μ^*.

8.2.1 Running Time of the Heuristic

The jobs are sorted in time $O(n \log n)$. In step 1, each $\hat{\eta}[p, p']$ is computed in time $O(n \log n)$ and, therefore, step 1 requires $O(n^3 \log n)$ time. The dynamic programming recursion in step 2 requires $O(n^2)$ time. An optimal batching policy is constructed in step 3 in $O(n)$ time. Step 4 requires solution of an assignment problem which runs in $O(n^3)$ time. Overall, the heuristic runs in $O(n^3 \log n)$ time.

8.3 A Lower Bounding Procedure

Each node of the branch and bound tree represents a unique partial batching policy, $\tilde{\mu}$. In the following we shall discuss a lower bounding procedure which is used to compute a lower bound, $\lambda(\tilde{\mu})$, on the total completion time of any schedule with a

complete batching policy compatible with $\tilde{\mu}$.

The contribution of job-order is minimized if the jobs are processed in the shortest total processing time order. Let ψ^* be the job-order such that $p < p' \Rightarrow (t_{1\psi^*(p)} + t_{2\psi^*(p)}) \leq (t_{1\psi^*(p')} + t_{2\psi^*(p')})$. Then

$$\lambda_0 = \eta(\psi^*) = \sum_{p=1}^{n} (n - p + 1)(t_{1\psi^*(p)} + t_{2\psi^*(p)})$$

is a lower bound on the contribution $\eta(\psi)$ of any job-order ψ.

Now, we shall discuss computation of a lower bound $\lambda_1[p, p']$ on $\eta[p, p', \psi]$ $\forall 1 \leq p \leq p' \leq n$. The quantity $\lambda_1[p, p']$ is used to compute a lower bound $\lambda_2[p, p']$ on $\eta[p, p']$. After computing $\lambda_2[p, p']$ $\forall 1 \leq p \leq p' \leq n$, we compute a lower bound $\lambda(\tilde{\mu})$ on the total completion time of any schedule with a complete batching policy compatible with $\tilde{\mu}$.

Computing $\lambda_1[p, p']$ reduces to a special type of assignment problem. For $1 \leq j \leq n$, $p \leq p'' \leq p'$, the contribution of assigning job j to position p'' is

$$\eta_{jp''} = t_{1j}(p'' - p). \tag{8.1}$$

Since jobs within a batch are processed in ascending order of t_{2j}, the quantity $\lambda_1[p, p']$ is obtained by assigning $(p' - p + 1)$ jobs to positions p ,..., p' so that:

1. job j is assigned to position $\hat{p}$ and job j' is assigned to position with $\tilde{p} > \hat{p}$, only if $t_{2j} \leq t_{2j'}$; and

2. the total contribution $\sum(\eta_{jp''} : j$ is assigned to p'', $p \leq p'' \leq p')$ of assigning jobs to positions p ,..., p' is minimized.

Hence, computing $\lambda_1[p, p']$ reduces to a special type of assignment problem.

We shall now show that such a problem can be solved by a dynamic programming recursion. We may assume, by relabelling if necessary, that $j < j' \Rightarrow t_{2j} \leq t_{2j'}$. Let $f_{jp''}$ = minimum total contribution when jobs j, ..., n are considered and assignments are made to positions p'', ..., p'. Observe that

$$\begin{array}{lcll} f_{np'} & = & \eta_{np'} & \\ f_{jp'} & = & \min\{\eta_{jp'}, f_{j(p'+1)}\} & \forall 1 \leq j \leq (n-1) \\ f_{np''} & = & \infty & \forall p \leq p'' \leq (p'-1) \\ f_{jp''} & = & \min\{\eta_{jp''} + f_{(j+1)(p''+1)}, \eta_{(j+1)p''}\} & \forall 1 \leq j \leq (n-1) \text{ and } p \leq p'' \leq (p'-1). \end{array}$$

The lower bound is

$$\lambda_1[p, p'] = f_{1p}.$$

After $\lambda_1[p, p']$ is obtained, a lower bound $\lambda_2[p, p']$ on $\eta[p, p']$ is obtained from the relation

$$\lambda_2[p, p'] = (s_1 + s_2)(n - p + 1) + \lambda_1[p, p].$$

The dynamic programming recursion discussed for the problem $1F2|s_i$,*fixed sequence*$|\sum C_j$ can now be used to obtain a lower bound on the contribution of a sequence of batches.

For $1 \leq p \leq n$, let $g(p)$ be a lower bound on the total contribution of the batches when some $(n-p+1)$ jobs are assigned to positions p, ..., n. Initialize $g(n) = 0$, and for each $1 \leq p \leq n$, compute

$$g(p) = \min_{p \leq p' \leq n} \{\lambda_2[p, p'] + g(p'+1)\}.$$

Consider any partial batching policy $\tilde{\mu} = (p_1, \ ..., \ p_k)$. If $p_k < (n-1)$, a lower bound on $\eta(\mu)$ is $\sum_{u=1}^{k} \lambda_2[p_{u-1}+1, p_u] + g(p_k + 1)$, where μ is any complete batching policy compatible with $\tilde{\mu}$. Hence, a lower bound $\lambda(\tilde{\mu})$ on the total completion time of any schedule with a complete batching policy compatible with $\tilde{\mu}$ is

$$\lambda(\tilde{\mu}) = \lambda_0 + \sum_{u=1}^{k} \lambda_2[p_{u-1}+1, p_u] + g(p_k + 1).$$

If $p_k = (n-1)$ or $p_k = n$, the problem of minimizing total completion time reduces to a problem $1F2|s_i$,*fixed batching policy*$|\sum C_j$ with a complete batching policy μ, where $\mu = (p_1, \ ..., \ p_k, n)$ if $p_k = (n-1)$ and $\mu = \tilde{\mu}$ if $p_k = n$. Each leaf node in the branch and bound tree corresponds to a complete batching policy.

8.3.1 Lowest and Highest Positions

For every job j we define the lowest position $L(j)$ and the highest position $H(j)$ such that if job j is assigned to position p, then $L(j) \leq p \leq H(j)$. If two jobs j and j' do not have identical processing times on both machines, j is called a *predecessor* of j' if $t_{1j} \leq t_{1j'}$ and $t_{2j} \leq t_{2j'}$. If $t_{1j} = t_{1j'}$ and $t_{2j} = t_{2j'}$ arbitrarily define one of j or j'

to be a predecessor of the other. Job j' is a *successor* of job j if j is a predecessor of j'. For each job j let $\nu_{pred}(j)$ be the number of predecessors and $\nu_{succ}(j)$ be the number of successors of j. There exists an optimal schedule in which for each job j, all predecessors of job j are completed before j and all successors of job j are completed after j. Hence, $L(j) \geq \nu_{pred}(j) + 1$ and $H(j) \leq n - \nu_{succ}(j)$.

Let $\eta^*_{j,p}(\psi)$ be the minimum contribution of job-order ψ given that $\psi(p) = j$. The quantity $\eta^*_{j,p}(\psi)$ is obtained by assigning job j to position p and processing the remaining jobs in the shortest total processing time order. If $\psi^*(p') = j$, then

$$\eta^*_{j,p}(\psi) = \begin{cases} \sum_{p''=1}^{p'-1}(n-p''+1)(t_{1\psi^*(p'')} + t_{2\psi^*(p'')}) \\ +\sum_{p''=p'}^{p-1}(n-p''+1)(t_{1\psi^*(p''+1)} + t_{2\psi^*(p''+1)}) \\ +(t_{1j}+t_{2j})(n-p+1) \\ +\sum_{p''=p+1}^{n}(n-p''+1)(t_{1\psi^*(p'')} + t_{2\psi^*(p'')}), & \text{if } p \geq p' \\ \\ \sum_{p''=1}^{p-1}(n-p''+1)(t_{1\psi^*(p'')} + t_{2\psi^*(p'')}) \\ +(t_{1j}+t_{2j})(n-p+1) \\ +\sum_{p''=p+1}^{p'}(n-p''+1)(t_{1\psi^*(p''-1)} + t_{2\psi^*(p''-1)}) \\ +\sum_{p''=p'+1}^{n}(n-p''+1)(t_{1\psi^*(p'')} + t_{2\psi^*(p'')}) & \text{if } p \leq p' \end{cases}$$

We use the quantities $\eta^*_{j,p}(\psi)$ to obtain the lowest and highest position of a job once we have an upper bound on the total completion time and a lower bound on the contribution of the batching policy.

Suppose that we have a schedule σ and a lower bound λ' on the contribution of the batching policy. In our search for a schedule better than σ we may assume that job j is not assigned to position p if $\eta^*_{j,p}(\psi) + \lambda' \geq \Delta(\sigma)$. For the least p such that $\eta^*_{j,p}(\psi) + \lambda' < \Delta(\sigma^*)$, we have $L(j) \geq p$. Hence, $L(j) = \max\{\nu_{pred}(j) + 1, p\}$.

Similarly, for the largest p such that $\eta^*_{j,p}(\psi) + \lambda' < \Delta(\sigma^*)$, we have $H(j) \leq p$. Hence, $H(j) = \min\{n - \nu_{succ}(j), p\}$.

We can now modify Equation 8.1. Since we want that job j be assigned to position p'' only if $L(j) \leq p'' \leq H(j)$, we set $\eta_{jp''} = \infty$ whenever $p'' < L(j)$ or $p'' > H(j)$. Hence, Equation 8.1 is modified as follows:

$$\eta_{jp''} = \begin{cases} t_{1j}(p'' - p) & \text{if } L(j) \leq p'' \leq H(j) \\ \infty & \text{otherwise} \end{cases} \tag{8.2}$$

First, we compute $L(j)$ and $H(j)$ using the initial upper bound obtained from the heuristic procedure and a lower bound $\lambda' = 0$ on the contribution of the batching policy. Then, the lower bounding procedure is used to obtain a lower bound $\lambda(\tilde{\mu})$, where $\tilde{\mu} = (0)$. A lower bound on the contribution of the batching policy is $\lambda' = \lambda(\tilde{\mu}) - \lambda_0$. Hence, we update $L(j)$ and $H(j)$ using the initial upper bound and the updated lower bound λ'on the contribution of the batching policy.

8.3.2 Running Time of the Lower Bounding Procedure

We compute all $L(j)$, $H(j)$, $\lambda_1[p, p']$, $\lambda_2[p, p']$ and $g(p)$ before processing any node in the branch and bound tree.

It takes $O(n \log n)$ time to sort the jobs in the shortest total processing time order and compute λ_0. Each $\eta^*_{j,p}(\psi)$ is computed in $O(n)$ time. Hence, it requires $O(n^3)$ time to compute all $\eta^*_{j,p}(\psi)$. All $\nu_{pred}(j)$ and $\nu_{succ}(j)$ are computed in $O(n^2)$ time.

Each $L(j)$ and $H(j)$ is computed in $O(n)$ time. Hence, it requires $O(n^2)$ time to compute all $L(j)$ and $H(j)$. Overall, it takes $O(n^3)$ time to get the initial lowest and highest positions. During the updating of the lowest and highest positions, quantities $\eta^*_{j,p}(\psi)$ are not computed again. Hence, the lowest and highest positions are updated in time $O(n^2)$.

Each $\eta_{jp''}$ is computed in constant time. For a given batch $[p, p']$ all $\eta_{jp''}$ is computed in time $O(n(p' - p)) \approx O(n^2)$. Each $f_{jp''}$ is computed in constant time. Hence, for batch $[p, p']$ all $f_{jp''}$ is computed in time $O(n(p' - p)) \approx O(n^2)$. Thus, $\lambda_1[p, p']$ is computed in time $O(n^2)$. The lower bound $\lambda_2[p, p']$ is obtained from $\lambda_1[p, p']$ in constant time. Since there are $O(n^2)$ batches, it takes $O(n^4)$ time to compute all $\lambda_2[p, p']$. Given all $\lambda_2[p, p']$, we compute all $g(p)$ in time $O(n^2)$. Hence, the above preprocessing requires $O(n^4)$ time.

An advantage of our lower bounding procedure is that most of the necessary computation is actually done before processing any node in the branch and bound tree. Each node in the branch and bound tree corresponds to a unique partial batching policy $\tilde{\mu} = (p_1, ..., p_k)$. If $p_k < (n - 1)$, we compute a lower bound $\lambda(\tilde{\mu})$. After the above preprocessing, we compute such a lower bound $\lambda(\tilde{\mu})$ in $O(n)$ time.

8.4 The Branch and Bound Scheme

Each node in the branch and bound tree represents a partial batching policy $\tilde{\mu}$. The root node represents the partial batching policy $\tilde{\mu} = (0)$ and each of the other nodes represents a partial batching policy of the type $\tilde{\mu} = (p_1, ..., p_k)$. The root node has n branches. The k-th branch is connected to a node that represents the partial batching policy (k). Each node representing a partial batching policy $\tilde{\mu} = (p_1, ..., p_k)$ with $p_k < (n-1)$ has $(n - p_k)$ branches $\hat{k} = 1, ..., (n - p_k)$. The $\hat{k}$-th branch is connected to a node that represents the partial batching policy $(p_1, ..., p_k, p_k + \hat{k})$.

If a node represents a partial batching policy $\tilde{\mu} = (p_1, ..., p_k)$ with $p_k = (n-1)$ or $p_k = n$, then the node is a leaf node and we solve the problem $1F2|s_i$,*fixed batching policy*$|\sum C_j$ with complete batching policy μ, where $\mu = (p_1, ..., p_k)$ if $p_k = n$ and $\mu = (p_1, ..., p_k, n)$ if $p_k = (n-1)$.

If a node is not a leaf node, we compute a lower bound. If the lower bound is less than the best solution, we generate all the branches. If the lower bound is equal to or greater than the best solution, we eliminate the node from further computation.

As we have discussed before, the heuristic requires $O(n^3 \log n)$ time and preprocessing requires $O(n^4)$ time. If a node is not a leaf node, we compute a lower bound in $O(n)$ time and if a node is a leaf node we solve a problem $1F2|s_i$,*fixed batching policy*$|\sum C_j$ in $O(n^3)$ time. The number of leaf nodes is 2^{n-1} and the number of non-leaf nodes is $O(2^{n-2})$. Hence, the worst case time complexity of the above branch

and bound scheme is $O(n^3 2^{n-1})$.

8.5 Alternate Algorithms

In the above, we outline a branch and bound algorithm which we call Algorithm A. An alternate algorithm, call it Algorithm B, is to generate a heuristic solution at each node processed. In our implementation of Algorithm B, we adopt the following heuristic at each node that represents a partial batching policy $\tilde{\mu} = (p_1, ..., p_k)$ with $p_k < (n\ -1)$: Let $k' = k$. While $p_{k'} \neq n$, set $p_{k'+1} = succ(p_{k'} + 1)$ and increase k' by 1. Solve the problem $1F2|s_i$,*fixed batching policy*$|\sum C_j$ with complete batching policy $\mu = (p_1, ..., p_k, ..., p_{k'})$.

Another alternate algorithm, call it Algorithm C, is to update all the lowest and highest positions each time a different lower bound is computed. As Algorithm A performs better than Algorithm B, a heuristic solution is not generated at each node processed by Algorithm C.

8.6 An Example

Our heuristic and lower bounding procedure can be given network representations similar to what we have discussed in Chapters 5 and 7. In this section we shall show the network representation by an example. We shall illustrate Algorithm A with the data shown in Example 7.3. For ease of reading we reproduce the example below.

Example 8.1 *Consider the following problem involving* $n = 3$ *jobs with* $s_1 = 2$ *and* $s_2 = 3$.

j	1	2	3
t_{1j}	8	9	7
t_{2j}	2	6	9
w_j	1	1	1

An interpretation of Steps 1, 2 and 3 of the heuristic can be given by a network as shown in Figure 8.1. There are three jobs 1, 2 and 3 to be assigned to three positions 1, 2 and 3. Introduce a dummy job 4 and assign it to a dummy position 4. Define a directed network $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ is the node-set and $\mathbf{E}$ is the arc-set. For each position p, there is a node p. For each pair of nodes p and p' with $1 \leq p < p' \leq (n+1)$, there is an arc $\langle p, p' \rangle$ which represents processing of batch $[p, p'-1]$ and is associated with weight $r(p, p') = \hat{\eta}[p, p'-1]$. In Steps 1, 2 and 3 a shortest path is computed from node 1 to $(n+1)$. The quantity $g(p)$ gives the length of the shortest path from node p. The arc weights and lengths of shortest paths from various nodes are shown in Figure 8.1.

As we can see from Figure 8.1, an optimal shortest path is $\langle \langle 1, 3 \rangle, \langle 3, 4 \rangle \rangle$. The path corresponds to batching policy $(2, 3)$ and the length 105 of the path is equal to the total completion time that we get by arranging the jobs in the ascending order of $(t_{1j} + t_{2j})$, using batching policy $(2, 3)$, and rearranging the jobs within batches in the ascending order of t_{2j}.

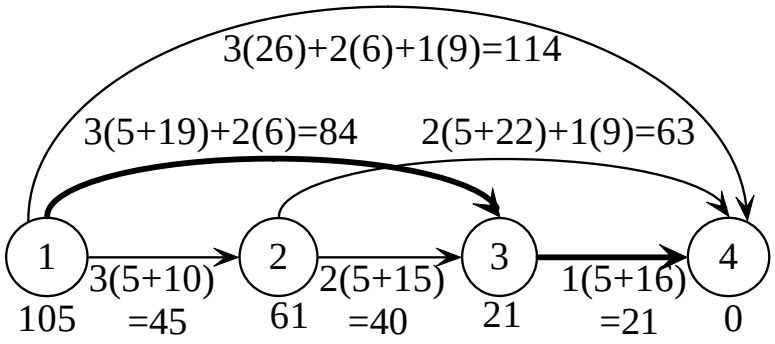

Figure 8.1: A network representation of heuristic steps 1,2,3

In Step 4 of the heuristic the problem $1F2|s_i$, *fixed batching policy*$|\sum C_j$ is solved with a batching policy $(2,3)$. In Section 7.3.2, the solution to Example 7.3 shows that the problem $1F2|s_i$, *fixed batching policy*$|\sum C_j$ with the data shown in Example 8.1 and batching policy $(2,3)$ has a solution with job-order (1,3,2) and the total completion time equal to 104. Thus, Step 4 yields an improved schedule.

The lower bound is computed in two stages. First, a lower bound λ_0 on the contribution of job-order is obtained as $\lambda_0 = 3(t_{11} + t_{21}) + 2(t_{12} + t_{22}) + 1(t_{13} + t_{23}) = 76$. The second stage can be given a network representation similar to what we have seen in the case of Steps 1, 2 and 3 of the heuristic. The nodes and arcs are defined in exactly the same way as we have done in the case of Steps 1, 2 and 3 of heuristic. However, the arc weights are computed differently. For all $1 \leq p < p' \leq (n+1)$, arc $\langle p, p' \rangle$ is associated with weight $r(p,p') = \lambda_2[p,p']$. The arc weights and lengths of shortest paths from various nodes are shown in Figure 8.2.

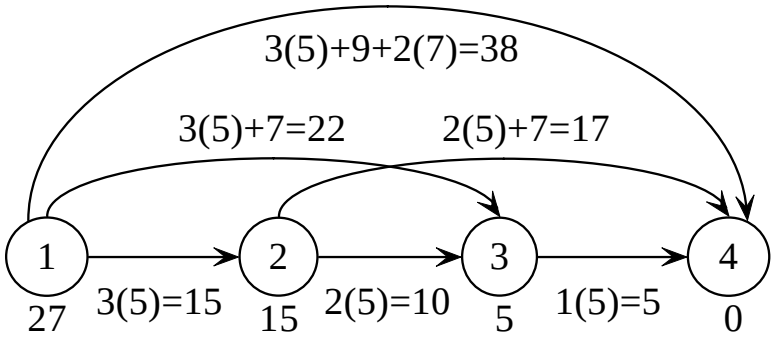

Figure 8.2: A network representation of the second stage of the lower bounding procedure

All the arc weights and shortest paths on the lower bounding graph (shown in Figure 8.2) are computed only once, before the start of the branch and bounding procedure. Given a partial or complete batching policy, $\tilde{\mu}$, we can compute a lower bound corresponding to $\tilde{\mu}$, using the lower bounding graph. An advantage of having the lower bounding graph is that in each node of the branch and bound procedure, we have a partial or complete batching policy and we can compute a lower bound corresponding to the batching policy with a few steps of computation. This is shown below.

Partial or Complete Batching Policy $\tilde{\mu}$	First Arc	Lower Bound
(0)	–	$27 + 76 = 103$
(1)	$\langle 1, 2 \rangle$	$15 + 15 + 76 = 106$
(2)	$\langle 1, 3 \rangle$	$22 + 5 + 76 = 103$
(3)	$\langle 1, 4 \rangle$	$38 + 76 = 114$

For example, consider $\tilde{\mu} = (1)$. Only those schedules are compatible to $\tilde{\mu} = (1)$ in which the jobs in the first position constitute a batch. From the weight on the arc $\langle 1, 2 \rangle$, we know that a lower bound on the contribution of a batch $[1, 1]$ is 15. From

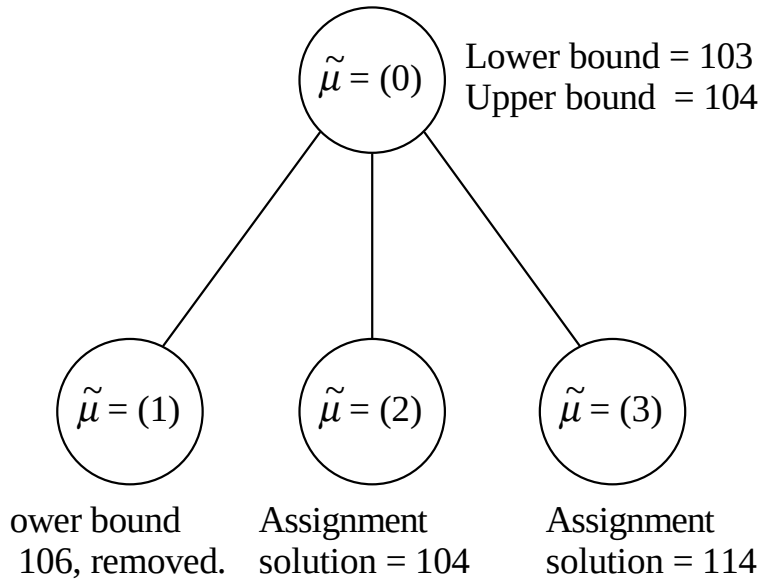

Figure 8.3: A branch and bound tree

the length of the shortest path from node 2, we know that the contribution of the remaining jobs and setups is 15. Now, adding the lower bound on the job-order, we get a lower bound 15+15+76=106 corresponding to $\tilde{\mu} = (1)$.

The above lower bounds are used in the branch and procedure. The branch and bound tree is shown in Figure 8.3. Observe that the node corresponding to $\tilde{\mu} = (2)$ yields the unique optimal solution (which is actually the heuristic solution). Hence, the optimal schedule is to use batching policy (2,3) and job-order (1,3,2) which gives the sequence of operations $(1,1)$, $(1,3)$, $(2,1)$, $(2,3)$, $(1,2)$, $(2,2)$ with a total completion time 104.

Note that in the above we do not use the concept of the high-low positions. If we use the concept of the high-low positions, a revised lower bound shows that the

heuristic solution is optimal and, therefore, the branch and bound procedure is not required to solve the problem with the above data. We omit the discussion here.

8.7 An IP Formulation

If a job j is assigned to position p as an l-th job within the batch, we say that the rank of job j or position p is l. We can define the contribution of assigning job j to position p as an l-th job within the batch so that the total completion time is obtained by summing up contribution of all assignments. If job j is assigned to position p as an l-th job within batch, operation $(1, j)$ precedes completion of $(n-p+l)$ jobs and operation $(2, j)$ is the last operation of job j and precedes completion of other $(n-p)$ jobs. Furthermore, if $l = 1$, a setup on machine 1 and another setup on machine 2 precedes the completion of $(n-p+1)$ jobs. Hence, the contribution of assigning job j to position p as an l-th job within a batch is

$$\eta_{jpl} = \begin{cases} (n-p+1)(t_{1j} + t_{2j} + s_1 + s_2) & \text{if } l = 1 \\ (n-p+l)t_{1j} + (n-p+1)t_{2j} & \text{if } l > 1 \end{cases}$$

For all $1 \leq j \leq n$ and $1 \leq l \leq p \leq n$ let $x_{jpl} = 1$ if job j is assigned to position p as an l-th job within batch and $x_{jpl} = 0$ otherwise. The following integer program solves the problem $1F2|s_i|\sum C_j$.

$$\begin{array}{lllll} \min & \sum_j \sum_p \sum_{l \leq p} x_{jpl}\eta_{jpl} & & & \\ \text{s.t.} & \sum_p \sum_{l \leq p} x_{jpl} & = & 1 & \forall j \\ & \sum_j \sum_{l \leq p} x_{jpl} & = & 1 & \forall p \\ & \sum_j x_{jpl} - \sum_j x_{j(p+1)1} - \sum_j x_{j(p+1)(l+1)} & \leq & 0 & \forall l \leq p < n \\ & \sum_j x_{jpl} - \sum_j x_{j(p+1)(l+1)} & \geq & 0 & \forall l \leq p < n \\ & x_{jpl} & \in & \{0,1\} & \forall j, p, l \end{array}$$

The first constraint ensures that each job is assigned to exactly one position and that each job has a unique rank. The second constraint ensures that each position is assigned to exactly one job and that each position has a unique rank. The third constraint ensures that if the rank of position p is l, then the rank of position $(p+1)$ is either 1 or $(l+1)$. The fourth constraint ensures that if the rank of position p is not l, then the rank of position $(p+1)$ is not $(l+1)$.

8.8 Computational Experience

Algorithms A, B and C and the Integer Program (IP) are tested on a randomly generated set of problems. For a discussion on generation of experimental data for machine scheduling problems, see Hall and Posner [56]. We generate problems with $n = 5$, 10, 15, 20, 25, 30 and 40. For each n, 10 problems are generated. We consider $s_i \sim U[2,4]$ and $t_{ij} \sim U[1,10]$, where U denotes the uniform integer distribution. Algorithms A, B and C are written in C and run on an AMD-K6-2/333 PC using Polaris. The C code for solving the assignment problem is obtained from an World Wide Web site maintained by MagicLogic Optimization Inc. [61]. The code is based on the methods described in Jonker and Volgenant [62]. The IP is coded in GAMS and solved using LAMPS run on an IBM RISC 6000 43P M140 using AIX.

For each of the Algorithms A, B and C we record number of nodes processed, number of leaf nodes processed, initial heuristic solution, initial lower bound, optimal

solution, and time in seconds. For the IP, time in seconds is recorded. The results are summarized in table 8.1. *Heuristic to optimal* is the ratio of heuristic solution to the optimal solution and *LB to optimal* is the ratio of initial lower bound to the optimal solution. More precisely, heuristic to optimal is $\Delta(\sigma)/\Delta(\sigma^*)$ and lower bound to optimal is $\lambda(\tilde{\mu})/\Delta(\sigma^*)$, where σ is the schedule generated by the heuristic in the first run, σ^* is an optimal schedule and $\tilde{\mu} = (0)$.

As Algorithm B updates the upper bound at each node, Algorithm B processes fewer nodes than Algorithm A. Similarly, as Algorithm C updates the lower bound on the contribution of batches at each node, Algorithm C processes fewer nodes than Algorithm A. However, in each case the advantage gained by eliminating nodes is outweighed by the increased amount of time required to update the upper or lower bounds. Algorithm A updates the lower bound at each node without updating the lower bound on the contribution of batches. In all cases, Algorithm A requires the least time. Between Algorithms B and C, Algorithm B requires less time.

The performance of the heuristic is encouraging. The average gap between the heuristic solution and the optimal solution is between 0% and 0.3%. The average gap between the initial lower bound and the optimal solution is 0% to 3.2%.

Algorithm A has been used to solve an instance of the problem with $n = 40$. The number of nodes, leaf nodes and time in seconds are 20 337 175, 11 369 524 and 20 803 respectively.

		Number of Jobs, n					
	Algorithm	5	10	15	20	25	30
Nodes	A	1.2	67.6	561.9	6 252	88 176	714 117
processed	B	1.2	66.8	560.4	6 217	87 710	711 351
	C	1.2	53.7	467.2	5 537	82 161	-
Leaf nodes	A	0.3	24.6	219.0	2 865	45 304	347 638
processed	B	0.3	23.8	217.3	2 846	45 034	346 088
	C	0.3	18.0	181.0	2 364	41 871	-
Time in	A	0.0	0.000	0.143	0.977	25.870	334.616
seconds	B	0.0	0.000	0.275	1.719	42.727	538.083
	C	0.0	0.000	0.824	35.673	1217.8	-
	IP	0.0	0.476	4.063	21.296	82.016	-
Heuristic to optimal		1	1.003	1.001	1.001	1.002	1.001
Lower bound to optimal		1	0.978	0.975	0.973	0.972	0.972

Table 8.1: Performance of algorithms for total completion time

8.9 Summary

In this chapter we consider the two-machine flow shop problem with the objective $\sum C_j$. The problem is open. However, we use our analysis on the fixed-sequence and fixed batching policy cases. We develop a heuristic, a branch and bound scheme and an integer programming formulation. We report computational experience for various implementations of the branch and bound scheme and the integer programming formulation.

The heuristic has four steps. In terms of the network representation of the problem $1F2|s_i$, *fixed sequence*$|\sum C_j$, Steps 1, 2 and 3 of the heuristic solve a shortest path problem. However, the computation of the arc weights is different from what is done

in the case of the problem $1F2|s_i$, *fixed sequence*$|\sum C_j$. Specifically, while we solve the problem $1F2|s_i$, *fixed sequence*$|\sum C_j$, we do not use the fact that in an optimal schedule jobs within a batch are processed in the ascending order of t_{2j} (because, we do not make any change to the given sequence). However, the fact is used in Steps 1, 2 and 3 of the heuristic. Step 4 of the heuristic solves a problem of the type $1F2|s_i$,*fixed batching policy*$|\sum C_j$ with the batching policy as obtained from Step 3 of the heuristic.

The lower bound is computed in two stages. First, a lower bound on the contribution of job-order is obtained by arranging the jobs in the ascending order of $(t_{1j}+t_{2j})$ and summing up the contribution of all jobs. The second stage can be given a network representation similar to the one for Steps 1, 2 and 3 of the heuristic. However, the arc weights are computed differently. Each arc represents a unique batch and the arc weight represents a lower bound on the contribution of the batch represented by the arc. The problem of computing each arc weight reduces to a special type of assignment problem. We develop an $O(n)$ time algorithm for this special type of assignment problem.

The branch and bound scheme essentially enumerates over all batching policies. Since there are at most 2^{n-1} batching policies and the fixed batching policy case is solved in $O(n^3)$ time, the branch and bound scheme requires $O(n^3 2^{n-1})$ time in the worst case.

We discuss three different implementations, A, B and C of the branch and bound scheme. In algorithm A we compute all the arc weights of the lower bounding graph only once, before the start of the branch and bound procedure. At each node of the branch and bound tree, we compute a different lower bound using the lower bounding graph. However, we do not revise the arc weights of the lower bounding procedure at each node. Also, we do not generate a new heuristic solution at each node.

In algorithm B, we generate a new heuristic solution at each node of the branch and bound tree. The performance of the algorithm is not better than that of Algorithm A. Therefore, we reject the idea of finding a new heuristic solution at each node. Algorithm C revises the arc weights of the lower bounding graph at each node. Neither the performance of algorithm C is better than that of algorithm A.

The advantage of algorithm A is, clearly, minimal computational requirement at each node. Algorithm A performs better than the integer program. Algorithm A has been used to solve problems with a maximum of 40 jobs.

The performance of the heuristic is encouraging. The average gap between the heuristic solution and the optimal solution is between 0% and 0.3%. The average gap between the initial lower bound and the optimal solution is 0% to 3.2%.

Chapter 9

Conclusion

Jobs go through various stages of operations. In a classical machine scheduling problem such as a flow shop, open shop and job shop problem, it is assumed that no single resource is used at two different stages. On the other hand, a flexible workforce or a versatile machine may be employed to perform various types of operations. Such resources may reduce the impact of uncertainties such as product mix changes and demand changes. Often these resources are associated with some setups that are required whenever a worker or machine switches from processing one type of operation to another, although several operations of the same type can be processed in succession after a single setup.

The presence of setups gives rise to the problem of choosing batch sizes that are neither too large nor too small. In the last one and a half decades, many researchers have addressed the problem of scheduling with setups. A majority of articles assume that there is only one type of scarce resource, which is typically a machine. Often

there can be two scarce resources such as a worker and a machine or a machine and a tool. The one-operator scheduling model considers a scheduling problem with setups and two scarce resources.

Santos [106, Chapter 4], Baker [12], Coffman et al. [30] and Julien [63] have pioneered the development on the problem of scheduling products that require multiple setups on a single facility. They have observed a number of dominance properties that restrict the search for an optimal schedule.

In Chapter 3, we show that the dominance properties apply to our problem. In the case of two machines, the properties imply that we can restrict the search for an optimal schedule to batching schedules. We also show by example that if the number of machines is greater than two, then we cannot restrict the search for an optimal schedule to batching schedules.

Psaraftis [97], Ahn and Hyun [3] and Ghosh [47] use similar dynamic programming recursions for various cases of the problem $1|s_{ii'}|\sum(w_j)C_j$. Ghosh and Gupta [48] extend the approach to the problem $1|s_{ii'}|L_{\max}$. In Chapter 4 we further extend the approach. We use the dominance properties discussed in Chapter 3 and show that many fixed-sequence cases can be solved using a dynamic programming approach similar to the ones used by Psaraftis [97], Ahn and Hyun [3] and Ghosh [47] and Ghosh and Gupta [48].

Psaraftis [97] points out that his approach applies to the classical traveling sales-

man problem. In the context of one-operator scheduling, the problem $1Om|s_{ii'}|\delta$ with a single job is equivalent to the m-city traveling salesman problem. The algorithm developed in Chapter 4 has polynomial time complexity for fixed m. The running time of the fixed-sequence cases with the objectives $L_{\max}$ and $\sum w_jC_j$ is $O(n^m)$ for fixed m. Hence, the algorithm may perform satisfactorily for small values of m.

There exist some strong relationships among the objectives $C_{\max}$, $L_{\max}$ and $\sum w_jC_j$. One such relationship is that the optimal sequence for each objective $C_{\max}$, $L_{\max}$ and $\sum w_jC_j$ is independent of the start time of the schedule. This has been pointed out in Chapter 4, and the relationship has been used to develop a common dynamic programming scheme for the fixed-sequence cases with these objectives. Another relationship is shown in Chapters 5 and 7. A similar network interpretation is provided for two-machine fixed-sequence cases with objectives $L_{\max}$ and $\sum w_jC_j$.

Recent development on the Monge-array algorithms by Aggarwal et al. [2], Wilber [122], Eppstein [37] and Galil and Park [40] speeds up dynamic programming recursions if Monge property is satisfied. In Chapter 7, we show that Monge-array algorithms can be used to improve running times of $1O2|s_i$, *fixed sequence*$|\sum w_jC_j$ and $1F2|s_i$, *fixed sequence*$|\sum w_jC_j$.

However, the Monge-array algorithms are not used in Chapter 5 to improve the running time of problems $1O2|s_i|L_{\max}$ and $1F2|s_i|L_{\max}$. Problems $1F2|s_i|L_{\max}$ and $1O2|s_i|L_{\max}$ are interpreted as shortest path problems, where the lengths of arcs and

paths are defined in a particular way. Two rules, the node elimination rule and the arc elimination rule are developed to speed up the computation of the "shortest path". By applying the node elimination rule and the arc elimination rule, we obtain simple $O(n)$ time algorithms for problems $1O2|s_i|L_{\max}$ and $1F2|s_i|L_{\max}$.

The node elimination rule applies to problems $1O2|s_i$, *fixed sequence*$|\sum w_jC_j$ and $1F2|s_i$, *fixed sequence*$|\sum w_jC_j$. However, the arc elimination rule does not apply to these problems. We remark here that if for some special structure of processing times and weights, arc elimination rule applies to problems $1O2|s_i$, *fixed sequence*$|\sum w_jC_j$ and $1F2|s_i$, *fixed sequence*$|\sum w_jC_j$, then algorithms similar to the ones discussed in Chapter 5 can be used to solve these problems.

As shown in Chapter 3, the problem $1|s_1, F = 1|L_{\max}$ is a special case of the problem $1F2|s_i|L_{\max}$. As in Chapter 5 we improve the running time of the problem $1F2|s_i|L_{\max}$ to $O(n)$, hence we get an improvement of the running time of the problem $1|s_1, F = 1|L_{\max}$ to $O(n)$.

In Chapter 6 we show that each of the problems $1F2|s_i|\sum w_jU_j$, $1O2|s_i|\sum w_jU_j$, $1F2|s_i|\sum U_j$, and $1O2|s_i|\sum U_j$ is $\mathcal{NP}$-hard. Lawler and Moore [77] and Sahni [104] give two pseudo-polynomial algorithms for the single machine scheduling problems with the objective $\sum w_jU_j$. A difference between these two algorithms is that Lawler and Moore [77] use processing time as a state variable and Sahni [104] uses weighted number of tardy jobs as a state variable. Hochbaum and Landy [58] and Brucker and

Kovalyov [24] extend the algorithms to the problem $1|s_1,\ F = 1|\sum w_jU_j$. Hochbaum and Landy [58] use processing time as a state variable and Brucker and Kovalyov [24] use weighted number of tardy jobs as a state variable.

We further extend the algorithms to problems $1F2|s_i|\sum w_jU_j$ and $1O2|s_i|\sum w_jU_j$. Algorithms of Sahni [104] and Brucker and Kovalyov [24] give polynomial-time approximation schemes. It is not known whether polynomial-time approximation schemes can be obtained for problems $1F2|s_i|\sum w_jU_j$ and $1O2|s_i|\sum w_jU_j$. Computational experience shows that it is better to consider weighted number of tardy jobs as a state variable as opposed to considering processing time as a state variable.

In Chapter 7 we show that the problem $1Fm|s_i,$ *fixed sequence*$|\sum w_jC_j$ is solved in $O(mn^3)$ time. However, it is not known whether the problem $1Om|s_i,$ *fixed sequence*$|\sum w_jC_j$ is solvable by an algorithm which is polynomial for variable m.

One motivation for analyzing the fixed-sequence cases is that in some cases, it is possible to obtain a job-order which dominates all the other job-orders. It follows from Theorem 4.1 that one such case is the problem with the objective $L_{\max}$, for which EDD order dominates the other job-orders. Theorem 4.2 points out another case with the objective $\sum w_jC_j$ and agreeable processing time and weight. The order 1, 2, ..., n dominates the other job-orders. Another motivation is to eventually obtain an enumeration scheme. However, there are $n!$ job-orders. We show in Chapter 7 that for $m = 2$ and $w_j = 1$ an alternate strategy is to enumerate over all batching

policies. The number of batching policies is 2^{n-1}. The problem with a fixed batching policy is solved in $O(n^3)$ time. However, such an enumeration scheme may not be useful for objective $\sum w_j C_j$. Both the problems $1O2|s_i$,*fixed batching policy*$|\sum w_j C_j$ and $1F2|s_i$,*fixed batching policy*$|\sum w_j C_j$ are $\mathcal{NP}$-hard.

In Chapter 8 we consider the problem $1F2|s_i|\sum C_j$. This problem is open. We discuss a heuristic and a lower bounding procedure which are used in a branch and bound scheme based on an enumeration over all batching policies as suggested in Chapter 7. We report performance of various implementations of the branch and bound scheme and an integer programming formulation.

Performance of the heuristic is encouraging. The average gap between the heuristic solution and the optimal solution is between 0% and 0.3%.. The average gap between the initial lower bound and the optimal solution is 0% to 3.2%. The branch and bound scheme generates an optimal solution within seconds for cases with up to 20 jobs. However, the required time grows very fast if $n \geq 30$. Problems with n as large as 40 have been solved by the branch and bound algorithm. The integer programming formulation has been used to solve problems with n as large as 28.

In Table 9.1 we summarize the running time of various solvable cases.

Problem	**Running Time**
$1Om\|s_{ii'}\|C_{\max}$	$O(m^2 2^m)$
$1Om\|s_i\|C_{\max}$	$O(1)$
$1Fm\|s_{ii'}\|C_{\max}$	$O(1)$
$1Om\|s_{ii'}\|L_{\max}$	$O(m^2(n+1)^m)$
$1Fm\|s_i\|L_{\max}$	$O(m(n+1)^m)$
$1Om\|s_{ii'}$,*fixed sequence*$\|\sum w_jC_j$	$O(m^2(n+1)^m)$
$1Fm\|s_i$,*fixed sequence*$\|\sum w_jC_j$	$O(mn^3)$
$1O2\|s_i\|L_{\max}$	$O(n)$
$1F2\|s_i\|L_{\max}$	$O(n)$
$1O2\|s_i\|\sum w_jU_j$	$O(n\min\{\max\{\sum t_{1j}, \sum t_{2j}\}, d_{\max}\}$ $\min\{\sum\sum(t_{ij}+ns_i), d_{\max}\})$
	$O(n\min\{\max\{\sum t_{1j}, \sum t_{2j}\}, d_{\max}\}\sum w_j)$
$1F2\|s_i\|\sum w_jU_j$	$O(n\min\{\sum t_{1j}, d_{\max}\}\min\{\sum\sum(t_{ij}+ns_i),$ $d_{\max}\})$
	$O(n\min\{\sum t_{1j}, d_{\max}\}\sum w_j)$
$1O2\|s_i$,*fixed sequence*$\|\sum w_jC_j$	$O(n)$
$1F2\|s_i$,*fixed sequence*$\|\sum w_jC_j$	$O(n)$
$1O2\|s_i$,*fixed batching policy*$\|\sum C_j$	$O(n^3)$
$1F2\|s_i$,*fixed batching policy*$\|\sum C_j$	$O(n^3)$

Table 9.1: Summary of running times

9.1 Future Research

There exist numerous possibilities for future research. As shown in Table 4.1, the classification of problems is not complete. For example, it follows from a result of Bruno and Downey [25] that the problem $1Om|s_i|L_{\max}$ is $\mathcal{NP}$-hard. However, it is not known whether this problem is strongly $\mathcal{NP}$-hard. Following are some other problems which needs to be classified:

- $1Om|s_i, \textit{fixed sequence}|\sum w_jC_j$
- $1Fm|s_{ii'}, \textit{fixed sequence}|\sum w_jC_j$
- $1Fm|s_{ii'}|L_{\max}$
- $1Fm|s_i|L_{\max}$
- $1O2|s_i|\sum w_jC_j$
- $1O2|s_i|\sum C_j$
- $1F2|s_i|\sum C_j$

As discussed above, further research is required to resolve the issue whether there exists a polynomial-time approximation scheme for problems $1F2|s_i|\sum w_jU_j$ and $1O2|s_i|\sum w_jU_j$.

We have shown that the arc elimination rule does not apply to problems $1O2|s_i$, *fixed sequence*$|\sum w_jC_j$ and $1F2|s_i$, *fixed sequence*$|\sum w_jC_j$. However, the arc elimination rule may be applicable for some special structure of the processing times and weights. If the arc elimination rule applies for some special structure of the processing times and weights, the running time may not be improved, but the algorithm may be simplified.

The heuristic given in Chapter 8 performs very well. One way to show a good performance of the heuristic is to find a worst case bound on the ratio of heuristic solution to optimal solution. We do not have an example in which the ratio is high. Hence, we conjecture that a good worst case bound on the ratio may be available through further research.

Finally, the model can be applied to a wider context if some of the assumptions are relaxed or modified. Here, we discuss only a few among a large number of possibilities. The assumption that $n_{ij} \geq 1$ can be relaxed. Sometimes a job may not require an operation on some machine. A versatile machine may be used for a large number of operations, and every job may require only a few types of operations.

Throughout we maintain an item availability assumption, which implies that completion time of a job is given by the completion time of the last operation of the job. However, if the jobs are moved in batches, a batch availability assumption is more appropriate. In such a case, the completion time of a job is the completion time of

the batch of operations that contains the last operation of the job.

As discussed by Błażewicz et al. [18, Chapter 10], some Flexible Manufacturing Systems (FMSs) are implemented using mainly one type of versatile machine. This type of FMS design can be represented by considering operators in parallel.

The dominance properties discussed in Chapter 3 are quite versatile. The properties can be extended to many extensions of our model. One exception is that of non-zero ready times. In presence of non-zero ready times, almost no property discussed in Chapter 3 holds. For example, suppose that a job j has two operations on a machine i, and as soon as one of the two operations is completed, a very important job arrives at the shop floor. It make sense to process the newly arrived job first (as it is very important) and then process the remaining operation of job j on machine i at some later time. Thus, the conclusion in Theorem 3.1 is violated, as the two operations of job j on machine i are not processed contiguously. Similarly, in the presence of non-zero ready times, job-orders may be different on various machines, a machine with positive inventory may be set up, and a machine may be switched before creating any inventory on the current machine.

Still, the properties discussed in Chapter 3 are versatile in the sense that the properties may be applicable with little modification to various cases including $n_{ij} \geq 0$, batch availability and parallel operators. One such modification required is in the case of $n_{ij} \geq 0$. We must change the condition stated in Theorem 3.5 that an inventory

be created before switching. Intuition suggests that if the current machine has an operation of a job that will leave the shop floor next, then it is wiser to complete the operation (and thus create an inventory) before switching to some other machine. However, if the current machine does not have any operation of a job that will leave the shop floor next, then an inventory may or may not be created before switching.

References

[1] J.O. Achugbue and F.Y. Chin. Scheduling the open shop to minimize mean flow time. *SIAM Journal of Computing*, 11:709–720, 1982.

[2] A. Aggarwal, M.M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987.

[3] B.-H. Ahn and J.-H. Hyun. Single facility multi-class job scheduling. *Computers & Operations Research*, 17(3):265–272, 1990.

[4] S. Albers and P. Brucker. The complexity of one-machine batching problems. *Discrete Applied Mathematics*, 47:87–107, 1993.

[5] A. Allahverdi, J.N.D. Gupta, and T. Aldowaisan. A review of scheduling research involving setup considerations. *Omega*, 27:219–239, 1999.

[6] G. Andrews. Personal conversation, 1999.

[7] Y.P. Aneja and N. Singh. Scheduling production of common components at a single facility. *IIE Transactions*, 22(3):234–237, 1990.

[8] Anonymous. Machine upgrades boost productivity. *Machine Design*, 67(4):40, 1995.

[9] Anonymous. CNC hobbing now standard at contract gearmaker. *Manufacturing Engineering*, 119(2):118–119, 1997.

[10] Anonymous. In-house production whips turnaround problem. *Manufacturing Engineering*, 120(2):86, 1998.

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

[12] K.R. Baker. Scheduling the production of components at a common facility. *IIE Transactions*, 20:32–35, 1988.

[13] D.D. Bedworth and J.E. Bailey. *Integrated Production Control Systems: Management, Analysis, Design.* Wiley, New York, 2nd edition, 1987.

[14] J. Błażewicz. Selected topics in scheduling theory. *Annals of Discrete Mathematics*, 31:1–60, 1987.

[15] J. Błażewicz, W. Cellary, R. Słowiński, and J. Węglarz. *Scheduling Under Resource Constraints: Deterministic Models.* J.C. Baltzer, Basel, Switzerland, 1986.

[16] J. Błażewicz, W. Domschke, and E. Pesch. The job shop scheduling problem: conventional and new solution techniques. *European Journal of Operational Research*, 93:1–33, 1996.

[17] J. Błażewicz, M. Dror, and J. Węglarz. Mathematical programming formulations for machine scheduling: a survey. *European Journal of Operational Research*, 51:283–300, 1991.

[18] J. Błażewicz, K. Ecker, E. Pesch, G. Schmidt, and J. Węglarz. *Scheduling in Computer and Manufacturing Processes.* Springer Verlag, Berlin and New York, 1996.

[19] J. Błażewicz, K. Ecker, G. Schmidt, and J. Węglarz. *Scheduling in Computer and Manufacturing Systems.* Springer Verlag, Berlin and New York, 1993.

[20] J. Błażewicz, W. Kubiak, and J. Szwarcfiter. Scheduling unit-time tasks on flow shops under resource constraints. *Annals of Operations Research*, 16:255–266, 1988.

[21] J. Błażewicz, J.K. Lenstra, and A.H.G. Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5:11–22, 1983.

[22] S. Brah, J. Hunsucker, and J. Shah. Mathematical modeling of scheduling problems. *Journal of Information & Optimization Sciences*, 12(1):113–137, 1991.

[23] P. Brucker. *Scheduling Algorithms.* Springer, Berlin, 1995.

[24] P. Brucker and M.Y. Kovalyov. Single machine batch scheduling to minimize the weighted number of late jobs. *Mathematical Methods of Operations Research*, 43:1–8, 1996.

[25] J. Bruno and P. Downey. Complexity of task sequencing with deadlines, setup times and changeover costs. *SIAM Journal on Computing*, 7:393–404, 1978.

[26] T.C.E. Cheng and G. Wang. A note on scheduling alternative operations in two-machine flow shops. *Journal of the Operational Reasearch Society*, 49:670–673, 1998.

[27] T.C.E. Cheng and G. Wang. Scheduling the fabrication and assembly of components in a two-machine flowshop. *IIE Transactions*, 31:135–143, 1999.

[28] P. Chrétienne, E.G. Coffman, Jr., J.K. Lenstra, and Z. Liu, editors. *Scheduling Theory and its Applications*. John Wiley & Sons Ltd, Chichester, 1995.

[29] E.G. Coffman, Jr., editor. *Computer & Job Shop Scheduling Theory*. Wiley, New York, 1976.

[30] E.G. Coffman, Jr., A. Nozari, and M. Yannakakis. Optimal scheduling of products with two subassemblies on a single machine. *Operations Research*, 37:426–436, 1989.

[31] E.G. Coffman, Jr., M. Yannakakis, M.J. Magazine, and C. Santos. Batch sizing and job sequencing on a single machine. *Annals of Operations Research*, 26:135–147, 1990.

[32] R.W. Conway, W.L. Maxwell, and L.W. Miller. *Theory of Scheduling*. Addison-Wesley Publishing Company, Inc., Reading, MA, 1967.

[33] S.A. Cook. The complexity of the theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM Press, New York, 1971.

[34] F.Y. Ding. A pairwise interchange solution procedure for a scheduling problem with production of components at a single facility. *Computers & Industrial Engineering*, 18:325–331, 1990.

[35] G. Dobson, U.S. Karmakar, and J.L. Rummel. Batching to minimize flow times on one machine. *Management Science*, 33(6):784–799, 1987.

[36] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[37] D. Eppstein. Sequence comparison with mixed convex and concave costs. *Journal of Algorithms*, 11:85–101, 1990.

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

[39] L.D. Frendall, S.A. Melnyk, and G. Ragatz. Information and scheduling in a dual resource constrained job shop. *International Journal of Production Research*, 34(10):2783–2802, 1996.

[40] Z. Galil and K. Park. A linear-time algorithm for concave one-dimensional dynamic programming. *Information Processing Letters*, 33:309–311, 1990.

[41] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. Freeman, San Francisco, 1979.

[42] M.R. Garey, D.S. Johnson, and R. Sethi. The complexity of flow-shop and job-shop scheduling. *Mathematics of Operations Research*, 1:117–129, 1976.

[43] R.S. Garfinkel. Motivation and modeling. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem*, chapter 2, pages 17–36. Wiley, 1985.

[44] A.E. Gerodimos, C.A. Glass, and C.N. Potts. Scheduling customized jobs on a single machine under item availability. Technical Report Report OR88, Faculty of Mathematics, University of Southampton, U.K., 1997.

[45] A.E. Gerodimos, C.A. Glass, and C.N. Potts. Scheduling the production of two-component jobs on a single machine. *European Journal of Operational Research*, to appear, 1999.

[46] A.E. Gerodimos, C.A. Glass, C.N. Potts, and T. Tautenhahn. Scheduling multi-operation jobs on a single machine. *Annals of Operations Research*, 92:87–105, 1999.

[47] J.B. Ghosh. Batch scheduling to minimize total completion time. *Operations Research Letters*, 16:271–275, 1994.

[48] J.B. Ghosh and J.N.D. Gupta. Batch scheduling to minimize maximum lateness. *Operations Research Letters*, 21:77–80, 1997.

[49] B. Gim and M.-H. Han. Economic scheduling of products with n components on a single machine. *European Journal of Operational Research*, 96:570–577, 1997.

[50] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the Association for Computer Machinery*, 23:665–679, 1976.

[51] 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 Mathematics*, 5:287–326, 1979.

[52] S.C. Graves. A review of production scheduling. *Operations Research*, 29:646–675, 1981.

[53] J.N.D. Gupta. Optimal schedules for single facility with two job classes. *Computers & Operations Research*, 11(4):409–413, 1984.

[54] J.N.D. Gupta. Single facility scheduling with multiple job classes. *European Journal of Operational Research*, 8:42–45, 1988.

[55] J.N.D. Gupta, J.C. Ho, and J.A.A. Van der Veen. Single machine hierarchical scheduling with customer orders and multiple job classes. *Annals of Operations Research*, 70:127–143, 1997.

[56] N.G. Hall and M.E. Posner. Generating experimental data for machine scheduling problems. *Operations Research*, to appear, 1999.

[57] M. Held and R.M. Karp. A dynamic programming approach to sequencing problems. *SIAM Journal of Applied Mathematics*, 10:196–210, 1962.

[58] D.S. Hochbaum and D. Landy. Scheduling with batching: minimizing the weighted number of tardy jobs. *Operations Research Letters*, 16:79–86, 1994.

[59] D.S. Hochbaum and D.E. Landy. Scheduling with batching: two job types. *Discrete Applied Mathematics*, 72:99–114, 1997.

[60] C. Hoskins. New gear cutting concept smashes speed/cost barriers. *Production*, 83(5):74, 1979.

[61] Magic Logic Optimization Inc. Linear assignment problem: Software download. http://www.magiclogic.com/assignment.html, 1999.

[62] R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38:325–340, 1987.

[63] F.M. Julien. *Scheduling Customer Orders*. PhD thesis, Department of Management Sciences, University of Waterloo, Canada, 1991.

[64] F.M. Julien and M.J. Magazine. Scheduling cusomer orders: an alternative production scheduling approach. *Journal of Manufacturing and Operations Management*, 3:177–199, 1990.

[65] B. Jurisch and W. Kubiak. Two-machine open shops with renewable resources. *Operations Research*, 45(4):544–552, 1997.

[66] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[67] H.V. Kher, M.J. Malhotra, P.R. Philipoom, and T.D. Fry. Modeling simultaneous worker learning and forgetting in dual resource consrtained systems. *European Journal of Operational Research*, 115:158–172, 1999.

[68] W. Kubiak. *Złożoność Obliczeniowa Algorytmów i Problemów Szeregowania Zadań przy Organiczeniach Zasobowych.* PhD thesis, ICS-Polish Academy of Sciences, Warsaw, 1988. in Polish.

[69] W. Kubiak, S.X.C. Lou, and Y. Wang. Mean flow time minimization in reentrant job shops with a hub. *Operations Research*, 44(5):764–776, 1996.

[70] D.E. Landy. *Batch scheduling for manufacturing.* PhD thesis, University of California, Berkeley, 1995.

[71] D.E. Lane and J.B. Sidney. Batching and scheduling in FMS hubs: flow time considerations. *Operations Research*, 41(6):1091–1103, 1993.

[72] E.L. Lawler. Recent results in the theory of machine scheduling. In A. Bachem, M. Grötschel, and B. Korte, editors, *Mathematical Programming: The State of the Art - Bonn 1982*, pages 202–234. Springer, Berlin, 1983.

[73] E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Minimizing maximum lateness in a two-machine open shop. *Mathematics of Operations Research*, 6:153–158, 1981. Erratum: [74].

[74] E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Erratum. *Mathematics of Operations Research*, 7:635, 1982.

[75] E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Recent developments in deterministic sequencing and scheduling: A survey. In M.A.H. Dempster, J.K. Lenstra, and A.H.G. Rinnooy Kan, editors, *Deterministic and Stochastic Scheduling*, pages 35–73. Reidel, Dordrecht, 1982.

[76] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and scheduling: algorithms and complexity. In S.C. Graves, A.H.G. Rinnooy Kan, and P. Zipkin, editors, *Handbooks in Operations Research and Management Science*, volume 4. Elsevier Science Publishers, 1993.

[77] E.L. Lawler and J.M. Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16:77–84, 1969.

[78] E.J. Lee and P.B. Mirchandini. Concurrent routing, sequencing, and setups for a two-machine flexible manufacturing cell. *IEEE Journal of Robotics and Automation*, 4(3):256–264, 1988.

[79] J.K. Lenstra. *Sequencing by Enumerative Methods.* Mathematical Centre Tract 69, Mathematical Centrum, Amsterdam, 1977.

[80] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[81] V. Lev and I. Adiri. V-shop scheduling. *European Journal of Operational Research*, 18:51–56, 1984.

[82] C.J. Liao and H.T. Lin. A case study in a dual resource constrained job shop. *International Journal of Production Research*, 36(11):3095–3111, 1998.

[83] P. Mellor. A review of job shop scheduling. *Operational Research Quarterly*, 17:161–171, 1966.

[84] J. Miltenburg. Balancing U-lines in a multiple U-line facility. *European Journal of Operational Research*, 109:1–23, 1998.

[85] C.L. Monma and C.N. Potts. On the complexity of scheduling with batch setup times. *Operations Research*, 37:798–804, 1989.

[86] J.M. Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15:102–109, 1968.

[87] T.E. Morton and D.W. Pentico. *Heuristic Scheduling Systems.* Wiley, New York, 1993.

[88] J.F. Muth and J.L. Thompson, editors. *Industrial Scheduling.* Englewood Cliffs, New Jersey, Prentice-Hall, 1963.

[89] D. Naddef and C. Santos. One-pass batching algorithms for the one-machine problem. *Discrete Applied Mathematics*, 21:133–145, 1988.

[90] C.H. Pan and J.S. Chen. Scheduling alternative operations in two-machine flow shops. *Journal of the Operationl Research Society*, 48:533–540, 1997.

[91] C.H. Papadimitriou. The Euclidean traveling salesman problem is $\mathcal{NP}$-complete. *Theoretical Computer Science*, 4:237–44, 1977.

[92] G. Paula. In-house gearing shortens turnaround. *Mechanical Engineering*, 120(3):30, 1998.

[93] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[94] C.N. Potts. Scheduling two job classes on a single machine. *Computers & Operations Research*, 18(5):411–415, 1991.

[95] C.N. Potts and M.Y. Kovalyov. Scheduling with batching: a review. *European Journal of Operational Research*, to appear, 1999.

[96] C.N. Potts and L.N. Van Wassenhove. Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. *Journal of the Operational Research Socity*, 43:395–406, 1992.

[97] H.N. Psaraftis. A dynamic programming approach for sequencing groups of identical jobs. *Operations Research*, 28(6):1347–1359, 1980.

[98] S.P. Rana and N. Singh. Group scheduling jobs on a single machine: a multi objective approach with preemptive priority structure. *European Journal of Operational Research*, 79:38–50, 1994.

[99] A.H.G. Rinnooy Kan. *Machine Scheduling Problems.* Martinus Nijhoff, The Hague, 1976.

[100] H. Röck. Some new results in no-wait flow shop scheduling. *Zeitschrift for Operations Research*, 28:1–16, 1984.

[101] V.K. Sahney. *Scheduling of Labor and Machine Limited Production Systems.* PhD thesis, University of Wisconsin, Madison, 1970.

[102] V.K. Sahney. Scheduling data transmission under an $\{S_i, 0\}$ policy. *Naval Research Logistics Quarterly*, 19:725–735, 1972.

[103] V.K. Sahney. Errata: single-server, two-machine sequencing with switching time. *Operations Research*, 22(5):1120, 1974.

[104] S.K. Sahni. Algorithms for scheduling independent tasks. *Journal of the Association for Computing Machinery*, 23(1):116–127, 1976.

[105] C. Santos. Batching and sequencing decisions under lead time considerations for single machine problems. Master's thesis, Department of Management Sciences, University of Waterloo, Canada, 1984.

[106] C. Santos. *Batching in Manufacturing Systems.* PhD thesis, Department of Management Sciences, University of Waterloo, Canada, 1988.

[107] C. Santos and M.J. Magazine. Batching in single operation manufacturing systems. *Operations Research Letters*, 4:99–103, 1985.

[108] D. Shallcross. A polynomial algorithm for a one machine batching problem. *Operations Research Letters*, 11:213–218, 1992.

[109] D. Sparling. Balancing just-in-time production units: the N U-line balancing problem. *INFOR*, 36(4):215–237, 1998.

[110] K.E. Stecke. Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems. *Management Science*, 29(3):273–288, 1983.

[111] C.S. Sung and C.K. Park. Scheduling of products with common and product-dependents components manufactured at a single facility. *Journal of the Operational Research Society*, 44(8):773–784, 1993.

[112] H. Süral, S. Kondakci, and N. Erkip. Scheduling unit-time tasks in renewable resource constrained flowshops. *ZOR-Methods and Models of Operations Research*, 36:497–516, 1992.

[113] V.S. Tanaev, V.S. Gorodn, and Y.M. Shafransky. *Scheduling Theory: Single-Stage Systems.* Kluwer Academic Publishers, Boston, 1994.

[114] T.L. Urban. Note: optimal balancing of U-shaped assembly lines. *Management Science*, 44(5):738–741, 1998.

[115] S. Van de Velde. *Machine Scheduling and Lagrangian Relaxation.* CWI, Amsterdam, 1991.

[116] G.S. Vasilash. A hobber for today. *Production*, 105(10):80, 1993.

[117] R.G. Vickson, M.J. Magazine, and C.A. Santos. Batching and sequencing of components at a single facility. *IIE Transactions*, 25:65–70, 1993.

[118] H.M. Wagner. *Principles of Operations Research.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1975.

[119] H.M. Wagner and T.M. Whitin. Dynamic version of the economic lot size model. *Management Science*, 5(1):89–96, 1958.

[120] M.Y. Wang, S.P. Sethi, and S.L. Van de Velde. Minimizing makespan in a class of reentrant shops. *Operations Research*, 45(5):702–712, 1997.

[121] S.T. Webster and K.R. Baker. Scheduling groups of jobs on a single machine. *Operations Research*, 43:692–703, 1995.

[122] R. Wilber. The concave least-weight subsequence problem revisited. *Journal of Algorithms*, 9(3):418–425, 1988.