Learning effect, Time-dependent Processing Time and Bicriteria Scheduling Problems in a Supply Chain

Jianbo Qian [1]
Operations Management Area
DeGroote School of Business
McMaster University, Hamilton, Ontario, Canada.

September 17, 2013

[1]e-mail: qianj2@mcmaster.ca

Abstract

This thesis contains two parts. In the first part, which contains Chapter 2 and Chapter 3, we consider scheduling problems with learning effect and time-dependent processing time on a single machine. In Chapter 2, we investigate the earliness-tardiness objective, as well as the objective without due date assignment consideration. By reducing them to a special linear assignment problem, we solve them in near-linear time. As a consequence, we improve the time complexity for some previous algorithms for scheduling problems with learning effect and/or time-dependent processing time. In Chapter 3, we investigate the total number of tardy jobs objective. By reducing them to a linear assignment problem, we solve them in polynomial time. For some important special cases, where there is only learning effect OR time-dependent processing time, we reduce the time complexity to quadratic time. In the second part, which contains Chapter 4 and Chapter 5, we investigate the bicriteria scheduling problems in a supply chain. We separate the objectives in two parts, where the delivery cost is one of them. We present efficient algorithms to identify all the Pareto-optimal solutions for various scenarios. In Chapter 4, we study the cases without due date assignment; while in Chapter 5 we study the cases with due date assignment consideration.

Acknowledgements

This thesis is dedicated to my daughter, Lily, and my wife, Youai, who are the main source of my motivation for work.

I would like to express my gratitude towards my supervisor Dr. George Steiner, who has been a strongly supportive advisor throughout my Ph.D study, who advised me on research topics as well as enriched my ideas while encouraging me to explore on my own and work independently. He has honed my research abilities, critical thinking and writing skills. I am especially grateful for his patience and the amount of time he spent reviewing many revisions of my work. His encouragement and faith in my abilities of research was the most important factor to complete this thesis.

I would like to thank Dr. Abad, Dr. Hassini and Dr. Baki for their insightful comments on my thesis. Their suggestions were crucial to the revision. I would also like to thank Dr. Abad, Dr. Hassini, Dr. Parlar and Dr. Baba for their excellent classes, from which I learned greatly, and their consistent help for my job and scholarship applications. The seminars organized by Dr. Hassini and Dr. Huang were also very helpful in broadening my research scope. Finally, I would like to thank Dr. Sandy Wu and Dr. Rui Zhang for their help during my study in McMaster University.

Contents

Keywords: Single-machine scheduling; learning effect; time-dependent, deteriorating effect; due date assignment; positional penalties; polynomial-time algorithm.

Chapter 1

Introduction and Literature Review

In this chapter, we introduce some important concepts and background of the topics that we will investigate in this report, as well as related previous research. Appendix A groups the notation that we use in this thesis.

Scheduling is the process of deciding how to allocate scarce resources such as equipment, labor and space to jobs, activities, tasks, or customers over time. It plays a crucial role in manufacturing and service industries, as well as information processing environments. In a typical scheduling problem, we are required to determine a schedule, which achieves certain objective(s) within accompanying constraints. Scheduling problems that we will discuss in this thesis are categorized as *deterministic* (as opposed to stochastic, where some data are random variables) *off-line* (as opposed to online, where some data are unknown beforehand) scheduling, where all data are deterministic and well-defined in advance.

1.1 Machine Scheduling

Inspired by the applications in production planning, deterministic off-line scheduling developed into a more specific research area, *deterministic off-line machine scheduling* (*machine scheduling*) from the late 1970s. It is concerned with assigning limited resources (called single or multiple machines) to a set of tasks (called jobs) to optimize a given objective function. In other words, jobs compete with each other for machine time. For comprehensive definitions, we refer readers to the textbooks [10] and [70].

1.1.1 Machine Scheduling Models

In this subsection, we introduce the classical notation system for machine scheduling models and related concepts that will be used in later chapters. In machine scheduling, models are classified

by the configuration of machines, the nature of jobs and the objective. By assumption, one machine can only process one job at a time and one job can only be executed by one machine at a time. In addition, a job can be started any time from the beginning (time 0) and can not be interrupted during its processing (called non-preemptive).

Now we outline the *three-field notation* system, $\alpha|\beta|\gamma$, which was established in [34] to describe a machine scheduling model:

- α indicates the configuration of machines, i.e., the type and the number of machines. $\alpha = 1$ specifies a single-machine environment. If there are multiple machines, it also refers to the type of machines. For instance, $\alpha = P2$ means a *parallel machine* environment consisting of two machines;

- β denotes the nature of jobs, i.e., the restrictions and the constraints of processing a job. For example, $\beta = pmtn$ implies that *preemption* is allowed such that a job can be interrupted during its processing and started over sometime later. We will put the due date assignment methods, such as CON, DIF, SLK (their meaning will be given later in this chapter) here. In particular, if β is left blank, this denotes the default setting, which means no restrictions or the constraints of processing a job.

- γ specifies the objective, which usually needs to be minimized. For instance, $\gamma = \sum w_j U_j$ means to minimize the weighted number of tardy jobs, where $U_j = 1$ if job j completes later than the *due date* d_j; otherwise $U_j = 0$, and w_j is the *tardiness penalty* (*weight*) of job j. We use (z, X) to denote a bicriteria Pareto-optimal objective, where z is the number of batches, X is the delivery cost and could be measured by total delivery time, maximum lateness, or total number of tardy jobs.

Using this notation system, the problem of *minimizing the weighted number of tardy jobs on a single machine* can be denoted by $1||\sum w_j U_j$. For more machine scheduling models and relevant results, we refer readers to the survey papers [51] and [14].

1.1.2 Algorithms in Machine Scheduling

The term *algorithm* is defined as a series of instructions for solving a given problem, in other words, a step-by-step procedure for calculations, or a precise rule (or set of rules) specifying how to solve certain problems. More precisely, an algorithm is an effective method expressed as a finite list of well-defined instructions for calculating a function. The fundamental issue of an algorithm is the *efficiency* for finding the optimal solution that is measured by *the maximum number of computational steps represented as a function of the input size of the problem in the*

worst case, termed the *running time*. The term *size* refers to the length of a problem's encoding. We outline two encoding forms by the following example. Integer "6" is encoded as "110" in the *binary* form and as "111111" in the *unary* form. Extending this example, a positive integer n is at most $\lfloor \log_2 n \rfloor + 1$ ones and zeros in binary encoding but exactly n ones in unary encoding.

These are central concepts in *computational complexity theory*, which is developed to study the nature of algorithmic tractability of problems and is widely used in and forms the foundation of computer science and combinatorial optimization. In computational complexity theory, the major concern about a problem is whether it is $\mathcal{NP}$-hard. Here the term "$\mathcal{NP}$" represents "*nondeterministic polynomial time*". A *decision problem* (as opposed to an optimization problem) is said to be in the class $\mathcal{NP}$, if its "Yes" answer (but not necessarily the No answer) can be verified by a reference algorithm in *polynomial time of its size under binary encoding* (*polynomial time*), where a decision problem is seeking either a "Yes" or "No" answer. Simply speaking, "a problem is $\mathcal{NP}$-hard" means that "a problem is at least as hard as the hardest decision problem in the class $\mathcal{NP}$ in terms of computational complexity". Another related concept is $\mathcal{NP}$-completeness. A decision problem is $\mathcal{NP}$-complete, if it is $\mathcal{NP}$-hard and is in the class $\mathcal{NP}$. The first $\mathcal{NP}$-complete problem given by [20] is the *satisfiability problem*, known as SAT: Is there a truth assignment for a given boolean formula? In his famous paper, Cook [20] proved that SAT is $\mathcal{NP}$-hard. This result is fundamental, because it provides an easier way to prove a problem's $\mathcal{NP}$-completeness: First the problem has to be shown to be in the class $\mathcal{NP}$, which is comparatively easy, and then a known $\mathcal{NP}$-complete problem is shown to be reducible to this problem in polynomial time. In contrast with the class $\mathcal{NP}$, the class $\mathcal{P}$, which stands for "*polynomial time*", contains all decision problems which can be solved by an algorithm in polynomial time. The question "*Is* $\mathcal{P} = \mathcal{NP}$*?*" is a one-million prize problem announced by the Clay Mathematics Institute of Cambridge, Massachusetts (CMI) in 2000 - http://www.claymath.org/millennium/P_vs_NP/ (July 29, 2008). In general, it is widely believed that $\mathcal{P} \neq \mathcal{NP}$ [21, 28, 43, 68, 89].

We now outline the categorizations of algorithms used in machine scheduling. From the point view of efficiency, all algorithms fall into the following three categories: *polynomial*, *pseudo-polynomial* and *nonefficient* (we use it for the moment, since there is no generally accepted term yet. It is usually exponential). Before giving the definitions, we introduce the "Big-O" notation. A (non-negative) function $T(n)$ is $O(f(n))$, iff there exists a constant c and a number X such that for all $n \geq X$, it is always true that $T(n) \leq cf(n)$. Given a problem, an algorithm to solve it is called polynomial if its running time $T(n) = O(n^k)$, where k is a constant and the size of the problem is polynomial in n under binary encoding. On the other hand, an algorithm is called pseudo-polynomial if its running time $T(n) = O(n^k)$, where k is a constant and $O(n)$ is the size of the problem under unary encoding. Any algorithm with longer (higher order) running time is called nonefficient. For example, an algorithm with running time $T(n) = 2^n$ is nonefficient,

where n is the size of problem under binary encoding. Recent developments on the details about nonefficient time algorithms in combinatorial optimization are reviewed by Woeginger [97].

Based on computational complexity theory, an $\mathcal{NP}$-hard problem is called $\mathcal{NP}$-hard only in the *ordinary* sense, iff it is solvable by a pseudo-polynomial algorithm. Otherwise, we call it *strongly* $\mathcal{NP}$-hard. From the following discussions, we will see their differences in the design and analysis of algorithms. Regarding the accuracy of solutions provided, algorithms are divided into three categories: *exact*, *approximate* and *heuristic*. An exact algorithm promises to deliver an optimal solution no matter how long it takes. A heuristic algorithm, however, only provides a solution fast, but there is no theoretical analysis for how good the solution is. It may be very far away from the optimum in the worst case. An approximation algorithm is between exact and heuristic algorithms. It provides a solution with a guaranteed approximation ratio to the optimum in the worst case. Suppose π^* is the optimal solution value for a minimization problem. A fully polynomial $(1+\varepsilon)$-approximation algorithm finds a solution value $\pi \leq (1+\varepsilon)\pi^*$ in polynomial time of problem size n, and $1/\varepsilon$, where ε could be any given positive value. For example, an algorithm that runs in $O(n^3/\epsilon)$ time is such an algorithm. Since a $(1+\varepsilon)$-approximation algorithm represents a series of algorithms for all $\varepsilon > 0$, it is called a "*fully polynomial time approximation scheme*" (*FPTAS*). The main advantage of an FPTAS is that one can get a solution arbitrarily close to the optimal solution in polynomial time. An FPTAS is about the trade-off between accuracy and efficiency. Other types of approximation algorithms can be found in the textbooks [43] and [89].

As mentioned before, there is no pseudo-polynomial algorithm for any strongly $\mathcal{NP}$-hard problem, unless $\mathcal{P} = \mathcal{NP}$. Therefore, finding a pseudo-polynomial algorithm is the main approach for establishing that a problem is $\mathcal{NP}$-hard in the ordinary sense. Due to the fact that "*the existence of an FPTAS for a problem implies the existence of a pseudo-polynomial algorithm as well*" [21], it is also impossible to find an FPTAS for a strongly $\mathcal{NP}$-hard problem, unless $\mathcal{P} = \mathcal{NP}$. For a problem which is $\mathcal{NP}$-hard in the ordinary sense, an FPTAS is the best possible theoretically. Since most problems in machine scheduling are $\mathcal{NP}$-hard, the methodology used in our thesis to study a problem will mainly go through the following stages. First, we will look for an $\mathcal{NP}$-hardness proof. If it can not be proven to be strongly $\mathcal{NP}$-hard at the first stage, we will try to develop a pseudo-polynomial algorithm to see if the problem is $\mathcal{NP}$-hard only in the ordinary sense. Once a pseudo-polynomial algorithm is obtained, we will try to convert it into an FPTAS.

1.2 Variable Processing Times

Traditionally, the processing times of jobs are fixed, regardless of their positions in the sequence or their starting time. In recent years, however, more and more researchers are investigating scheduling problems with variable processing times of two types: learning effect and time-dependent processing times.

Learning effect As early as in the 1930s, Wright [98] noticed that in the aircraft industry the working costs per unit decreased with an increasing production output. He formulated the so-called 80% hypothesis, stating that with every redoubling of output the unit processing time decreases by 20%. This learning effect has huge impact in mass production. For example, it was reported that in the semiconductor industry, the efficiency gains cause the price to drop by 10-30% [95]. Recently, a new proof of the impact of learning effect from a new car-assembly plant was presented in [77].

Learning effects are important for production problems involving a significant level of human activities, such as machine setup, machine cleaning, machine operating and controlling, machine maintenance, machine failure removal, machine data reading/understanding/interpretation, and all kinds of handwork. They are especially important when the production environment changes. Such changes include, among others, new workers, investments in new machines or replacement of equipment, workflow changes, and the acceptance of new jobs. Even small changes to the production environment like a software update, a new design of the format of important documents, a new organization of the spare parts depot, cause learning effects. Every time a worker needs to get used to a new circumstance a learning experience will occur. Since there are usually high levels of such human involvements in scheduling environment which changes frequently, and these activities are subject to learning, it is reasonable to consider learning effects in scheduling problems [5].

Biskup [5] was the first to consider the learning effect in scheduling problems. He proposed the following model:

$$p_{[j]} = a_{[j]} j^c, \tag{1.1}$$

where $p_{[j]}$ and $a_{[j]}$, respectively, represent the actual processing time and "normal" processing time (i.e., without learning effect) of the job at the jth position in the schedule, and $c \leq 0$ represents the rate of learning. It is easy to verify that if $c = \log_2 0.8 = -0.322$, it corresponds to the aforementioned 80% hypothesis. This effect is shown in Fig. 1 [6], where the x-axis represents the number of products produced, and the y-axis represents the processing time needed.

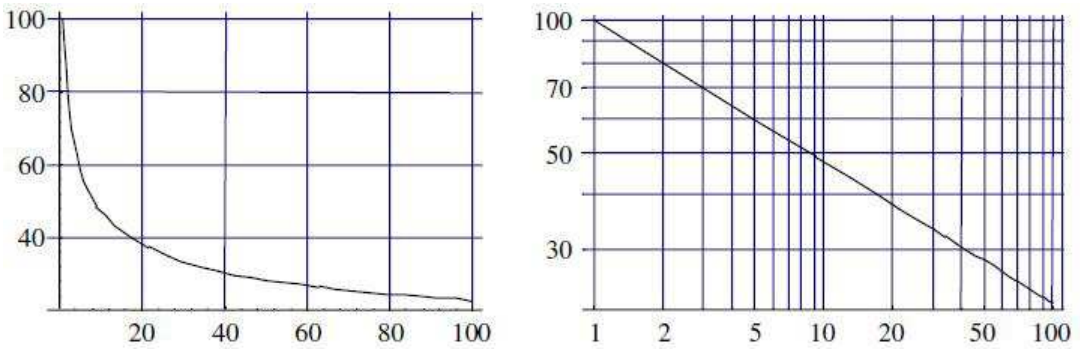

Figure 1.1: The learning curve with a learning rate of 80% depicted in a normal and a double-logarithmic coordinate system

If $c > 0$, then the model captures the effect of process deterioration [33]. Most researchers used an equivalent model $p_{ir} = p_i r^c$, where r represents the position. Mosheiov [62] investigated scheduling problems with learning effects described by (1.1) on parallel machines and Mosheiov et al.[63] studied more general learning functions. A comprehensive survey about the learning effect in scheduling problems can be found in [6]. Further reviews of scheduling models with positional effects and learning were presented in [45, 46, 75].

Time-dependent processing time On the other hand, in some other practical situations, the actual processing time of a job depends on its starting time. For example, Gupta et al. [36] modeled a multiple loan repayment problem in financial management as this type of scheduling problem. In steel production, steel ingots are to be heated to the required temperature before rolling can begin. The time taken to heat depends on the size and current temperature of the ingot, which further depends on the amount of time the ingot has been waiting to be processed. Similarly, in fire fighting, the time required to control a fire is longer if the start of the fire fighting effort is delayed. Some other empirical applications exist in hospital emergency room scheduling and crime scene response scheduling for police [50].

This concept of time-dependent processing times was first introduced by Browne and Yechiali [7]. The most commonly used model is

$$p_{[j]} = a_{[j]} + bS_{[j]}, \tag{1.2}$$

where $a_{[j]}$ also denotes the "normal" processing time, b is a constant and $S_{[j]}$ denotes the starting time of the job in the jth position. If $b > 0$, then it reflects deteriorating processing time. Mosheiov presented a V-shape policy and a $\wedge$-shape policy, respectively, in [59] and [60] for scheduling deteriorating jobs. Some researchers used position-dependent b_j [18]. However, the problem usually becomes significantly harder in most of those cases. Gawiejnowicz gives a

comprehensive exposition of time-dependent models [29]. Cheng et al. also presented a concise survey [18] for these problems.

These two types of variable processing time are usually considered separately. Lee [54] was the first to consider them simultaneously. Wang [90] proposed the following model:

$$p_{[j]} = (a_{[j]} + bS_{[j]})j^c, \tag{1.3}$$

where $p_{[j]}$, $S_{[j]}$, b and $c \leq 0$ all have the same meaning as before. It was followed by Wang [92], Wang and Cheng [94], Wang et al. [91], Wang and Cheng [93].

This model reflects the following practical scenario: an operator obtains additional skills by learning from experience, at the same time, the machine that he/she operates is subjected to wear and tear, i.e., deteriorates with time [75]. A recent review by Janiak et al. focused on a variety of models with this type of combined effects [46].

We can see that each of the above discussed cases of variable processing time is a special case of this general model. If $b = 0$, we have the learning effect problem; if $c = 0$, we have the time-dependent processing time problem.

1.3 Supply Chain Scheduling

Supply chain management has been one of the most important topics in operations management, in both the manufacturing and the service sectors, since the eighties. Most of the supply chain literature focuses on issues on the strategic level, using stochastic models. A recent survey paper [88], however, suggested that over 11% of the U.S. Gross National Product is spent on logistics, and for many products, the logistics costs more than 30% of the cost of goods sold. Therefore, to improve the overall operational performance, it is necessary to study scheduling models which consider inbound production and outbound deliveries simultaneously. Our research deals with supply chain problems on the operational level, using deterministic models. This type of scheduling was named *supply chain scheduling* by Hall and Potts [38].

In contrast with classical machine scheduling problems, in supply chain scheduling problems, deliveries are also considered part of tasks. In other words, there are two decisions to be made: How to process jobs on machines and how to deliver jobs to customers? Due to the fact that delivering jobs in batches saves delivery costs and setup time, the delivery operation does bring a new question: *How to group jobs in batches for both production and deliveries*? This connects supply chain scheduling to *batch scheduling*, which is a well-studied research area in machine scheduling. They are different, however, as delivery costs are not part of the usual objectives in batch scheduling. Detailed discussions about batch scheduling can be found in the survey

paper [72]. From the view-point of computational complexity, how to group jobs in batches does make some easy problems harder. For example, the problem of *minimizing the total weighted completion time on a single machine* can be easily solved by sequencing all jobs in the *weighted shortest processing time* order in polynomial time [70], but its supply chain scheduling counterpart of *minimizing the sum of the total weighted completion times and the batch-delivery costs on a single machine* is strongly $\mathcal{NP}$-hard [38].

Extending the β field, the three-field notation system can still be applicable in supply chain scheduling. For instance, $1|\Delta|\sum w_jU_j + zq$ indicates a supply chain scheduling model on a single machine with the goal of minimizing the sum of the weighted number of tardy jobs and the batch-delivery costs, where Δ is the machine time needed to set up a new batch, q is the delivery cost per batch and z is the number of batches.

1.3.1 Supply Chain Scheduling Models with Delivery Costs

In supply chain scheduling problems, the delivery costs are usually related to batching. Let us look at this important concept:

Batching Batching is to decide how to schedule certain jobs consecutively. It is another important research issue in scheduling theory and has a lot of practical motivations. For example, in the chemical, food processing and fertilizer industries, engineering parts are divided into different families such that machine setups are not needed between the production of parts in the same family, using the principles of group technology [41]. In flow shops and other multistage production systems, machine utilization may be improved by the batching of products for transportation between machines where the transportation device is a bottleneck. There are also applications where multiple jobs are naturally processed simultaneously in a batch, for example the burn-in testing operation in semiconductor manufacturing [52] (such scenarios were mostly investigated in the batching machine literature, which have different nature from our research). When an order consists of many similar items for the same customer, batching just the part of the order for delivery to this customer improves customer service, since the customer does not have to wait until the entire order has been processed [39].

Potts and Kovalyov introduced the scheduling models that take into account logistics [71]. However, they considered only delivery time, not including delivery costs, in their model. Cheng and Kahlbacher [11] were the first to study a machine scheduling model with delivery costs consideration. Lee and Chen [53] further extended this model in [11] into one with a limited delivery capacity.

Hall and Potts [38] were the first to introduce delivery costs into the objective, which can

be minimizing total delivery time, minimizing maximum lateness, or minimizing the weighted number of tardy jobs on a single machine. In their models, however, they made the assumption that tardy jobs are not delivered to customers. With this assumption, they were able to present a pseudo-polynomial algorithm, either when jobs are for a single customer, or for a fixed number of customers. Their algorithm becomes polynomial when all jobs have equal weights. Their model, however, did not consider batch-setup time, which consumes machine time in most real cases. Recently, Steiner and Zhang [85] investigate the problem in a model in which the tardy jobs also get delivered.

In the same paper [38], Hall and Potts also studied other forms of objectives to minimize the total scheduling and delivery costs. As it is a fairly new research area, there are relatively few papers dealing specifically with scheduling problems in supply chains. Chen and Hall [16] extended these problems to supply chains with assembly-type manufacturing systems. Dawande et al. [22] extended them to distribution systems. Some of the issues studied in these papers are related to previous work on coordinating production and distribution systems. We mention here the paper by Williams [96], and the one by Lee and Chen [53], where they consider the integration of transportation time and capacity issues with scheduling decisions. Li et al. [56] studied the problem of minimizing the average job-arrival time, which includes the travel time to the customers. Chen and Vairaktarakis [15] presented polynomial time solutions for the problem of minimizing a convex combination of the mean arrival times and the total distribution cost, where the latter includes fixed delivery costs and variable costs dependent on the delivery routes, with a fixed number of customers. Pundoor and Chen [73] studied a model where the objective is to minimize a linear convex combination of the maximum delivery tardiness and total delivery costs. Selvarajah and Steiner [79, 80, 81] developed exact and approximation algorithms for the supplier's problem of minimizing the sum of the total weighted completion time and batch-delivery costs. Agnetis et al. [2] investigated the problem of rescheduling to resolve conflicts between the supplier's and the manufacturers' ideal schedules. Hall and Potts [39] studied the coordination of scheduling and batch deliveries with various scheduling objectives, such as minimizing total delivery time, minimizing maximum lateness, or minimizing the weighted number of tardy jobs. Moreover, Hall et al. [40] edited a special issue focusing on the area of supply chain coordination and scheduling, where Tang et al. [87] considered a production and distribution model taking into account inventory control issues with one supplier and multiple buyers. Manoj et al. [58] studied the decentralized and joint optimization problems of a model with one manufacturer, one distributor, and several retailers in a just-in-time environment. The most recent comprehensive survey paper was by Chen [17].

1.3.2 Supply Chain Scheduling Models with Delivery Costs and Due Date Assignment

Before our literature review on this subject, let's first introduce some concepts and background for due date assignment.

Due date assignment In a flexible scheduling environment, the attainability of due dates and the feasibility of schedules are both taken into consideration at the same time. Taking advantage of this added flexibility may lead to improved overall system performance.

Meeting due dates has always been one of the most important objectives in scheduling and supply chain management. Customers usually require that suppliers either meet *contracted due dates* or pay large penalties. For example, [84] reported that the tardiness penalties in aerospace industries may be as high as one million dollars per day for suppliers of aircraft components.

While traditional scheduling models considered due dates as given by exogenous decisions (see Baker and Scudder [4]), in a more flexible and integrated system, they are determined by taking into account the system's ability to meet the quoted delivery dates. For this reason, numerous recent studies have viewed due date assignment as part of the scheduling process and showed how the ability to control due dates can be a major factor in improving system performance. In order to avoid tardiness penalties, companies are under increasing pressure to quote *attainable* delivery dates for customer orders. Naturally, longer due dates are easier to meet, but promising delivery dates too far into the future may not be acceptable to the customer. At the same time, shorter due dates increase the probability that the order will be delivered late. Thus there is an important tradeoff between assigning relatively short due dates to customer orders and avoiding tardiness penalties, which could be very substantial (see e.g.[84]). This creates the need for a methodology that allows firms to quote attainable delivery dates and obtain efficient schedules at the same time.

Many different due date assignment methods have been studied in the literature (see [30, 31, 32, 47] for extensive surveys). We briefly review four of the more commonly used methods below:

- The *common* due date assignment method (referred to as *CON* in short), in which all jobs are assigned the same due date, that is $d_j = d$ for $j = 1, ..., n$, where d_j denotes the due date of job j and $d \geq 0$ is a decision variable (see Panwalkar *et al.* [67]).

- The *common due window* assignment method (we will refer to this method as *CONW*), where the scheduler can assign a single desired time window, $[\underline{d}, \overline{d} = \underline{d} + D]$, for the completion time of each job, and the objective includes a linear penalty for both $\underline{d}$ and D

(see Liman *et al.* [57]). It is easy to observe that the *CON* method is a special case of the *CONW* method when the penalty for the window length (D) is large enough.

- The *slack* due date assignment method (usually referred to as *SLK*), in which all jobs are given a flow allowance that reflects equal waiting time (equal slacks), that is, $d_j = p_j + s$ for $j = 1, ..., n$, where p_j is the processing time of job j and $s \geq 0$ is a decision variable (see Adamopoulos and Pappis [1]).

- The *unrestricted* due date assignment method (usually referred to as *DIF*), in which each job can be assigned a different due date with no restrictions (see Seidmann *et al.* [78]).

Cheng and Kovalyov [12] studied a batch scheduling model which took into account due-date-assignment costs. Their objective is to minimize the number of tardy jobs with batch-setup time and a uniform assignable due date on grouped jobs, where jobs are divided into different groups and jobs in the same group are identical. Changing the processing of jobs from one group to another requires a sequence-independent batch-setup time, which depends on the groups. They first developed a pseudo-polynomial algorithm for it and then converted it into an FPTAS. Their model, however, does not consider batch-delivery costs as part of the objective.

A more complex model, with distinct assignable due dates, was studied by Shabtay and Steiner [82]. In their model, each job has a contracted due date, and each job can be assigned an arbitrary extended due date. The goal is to minimize the sum of the due-date-assignment costs and the weighted number of tardy jobs with respect to the assigned due dates. They first gave a strong $\mathcal{NP}$-hardness proof for the general case and then presented two polynomial algorithms for two important special cases: one with equal due-date-assignment costs and zero contracted due dates and the other one with equal due-date-assignment costs, zero contracted due dates and equal tardy penalties. Their model, however, does not include batching or delivery costs either.

There are some researchers who considered optimizing other scheduling objectives in this context. Chen [13] studied a single-machine scheduling problem, where the objective is to minimize the sum of earliness and tardiness penalties and delivery costs with a common assigned due date. Yang [99] focused only on tardiness penalties with quoted delivery dates but without delivery costs. In his paper, two problems are studied: one to minimize the total batch earliness and the other one to minimize the largest batch earliness.

1.4 Bicriteria scheduling

Before the 1990s, the objective function in most scheduling problems contains only one performance criterion. However, in many practical situations, quality is a multicriteria notion by

nature. For example, schedulers often need to meet the due date deadline in keeping high level of customer service level on one hand, and to reduce various costs, such as holding cost and transportation cost, on the other hand. These criteria almost always conflict with each other. If only one criterion is taken into account, then the outcome tends to be unbalanced and biased, no matter what criterion is considered. For example, if all the resources are set on keeping the costs low, then the service level is likely to be low too, and vice versa. In order to reach an acceptable compromise, it is necessary to measure the quality of a solution on all important criteria. This observation has led to the development of the area of multicriteria scheduling.

Since objectives with two criteria are more common in practice and easier to tackle, we consider only bicriteria scheduling problems. Suppose that f and g are the two performance criteria that we want to take into account. Without loss of generality, we assume that these criteria are to be minimized. Very likely there will be no schedule that minimizes both performance criteria simultaneously, which implies that it is almost certain that we have to sacrifice on the quality of one criterion. If f is far more important than g, then a natural choice is to find the optimum value of f, and choose from among optimum schedules for f the one that performs best on g. Such an approach is called *hierarchical optimization* or *lexicographical optimization*.

Lexicographical optimization is probably inappropriate if no criterion is preferable to the other, since it may lead to an unbalanced schedule; or the second criterion can be greatly improved at very little cost in the first criterion. In such cases, simultaneous optimization is a better choice. Evans [26] and Fry et al. [27] distinguish three different approaches in simultaneous optimization: a priori optimization, interactive optimization, and a posteriori optimization. In a priori optimization, both criteria are integrated into one composite objective function $F(f, g)$ for some given function F, then an optimum solution is determined for this function as a whole. F is usually linear, for example $af + bg$, where a and b are given constants that indicate the relative importance of criterion f and criterion g [44].

This approach has two shortcomings. The first one comes from practice: it is usually hard to know what this function F looks like. The scheduler may find it hard to express the preference in a function F. The second drawback is that it is often hard to minimize the function $F(f, g)$ directly. There are two ways to avoid these drawbacks. The first one requires the scheduler's active involvement during the solution process. Given some solutions, the decision maker must indicate which one is better, and if not satisfied yet, in which direction the search should continue; this scenario is called interactive optimization. The second option is to solve the problem in a roundabout way. The idea behind it is that we select from the set of solutions a subset that contains an optimum solution for each reasonable composite objective function F that we can think of. If the function F is known, then we compute the optimum solution in this set. If F is not known, then we present this set to the decision maker and let him/her choose; this scenario is

called a posteriori optimization. Note that a posteriori optimization is clearly the most difficult variant of the three approaches; if we can solve this problem, then we can solve the other two as well, if we may assume that the function F specified by the decision maker beforehand in a priori optimization will be reasonable. Therefore, we assume from now on that we consider the a posteriori variant when we write simultaneous optimization without further specification.

Chapter 2

Fast Algorithms for Scheduling with Learning Effects and Time-dependent Processing Times on a Single Machine

The general problem we study may be stated as follows: n independent, non-preemptive jobs, $J = \{1, 2, ..., n\}$, are available for processing at time zero and are to be processed on a single machine. A schedule is defined by a job sequence $\pi = ([1], [2], ..., [n])$, where $[j]$ represents the job that is in the jth position in π for $j = 1, 2, ..., n$. Our objective is to determine a schedule which minimizes a general unified cost function that is the sum of scheduling costs, that can be expressed by using positional penalties. This unified cost function can be formulated as follows :

$$g(\pi) = \sum_{i=1}^{n} \xi_i \eta_{[i]}, \tag{2.1}$$

where ξ_i is a *positional, job-independent penalty* for *any* job scheduled in the ith position for $i = 1, ..., n$.

Our model is an adaptation from [55], where Leyvand et al. proposed a unified framework for scheduling problems with controllable processing time, and solved it using an algorithm for general linear assignment in $O(n^3)$.

The chapter is organized as follows. In Subsection 2.1, we present an $O(n \log n)$ optimization algorithm to determine the optimal schedule for the $1 \,||\, \sum \xi_i \eta_{[i]}$ unified problem. In the next subsection, we present two groups of scheduling problems. In the first group, the objective function includes costs for earliness, tardiness, makespan, and due date assignment. The second group contains some additional classical scheduling problems. These include minimizing the makespan, the sum of completion times, the variation of job completion times and the variation of waiting times with controllable processing times. We show that, for four different due date

assignment methods and all scheduling problems without due date consideration, the objective can be reformulated as a special case of (2.1), which enables us to solve this set of problems in $O(n \log n)$ time. The last subsection contains a summary and our concluding remarks.

2.1 Assignment Problems on Permuted Monge Matrices

First we introduce two special types of matrices:

Definition 1 *A matrix $C = (c_{ij})_{m\times n}$ is called a* **Monge matrix** *if it fulfills the following so-called Monge property:*

$$c_{ij} + c_{rs} \leq c_{is} + c_{rj}$$

for all $1 \leq i < r \leq m$ and $1 \leq j < s \leq n$. C is said to be a **permuted Monge matrix** *if its rows and columns can be permuted respectively such that the resulting matrix is a Monge matrix.*

Definition 2 *A matrix $C = (c_{ij})_{m\times n}$ is called a* **product matrix** *if there exist two non-negative, real vectors $\mathbf{a} = (a_1, a_2, ..., a_m)$ and $\mathbf{b} = (b_1, b_2, ..., b_n)$, such that $c_{ij} = a_i b_j$.*

The following proposition establishes their relations:

Lemma 1 *Every product matrix is a permuted Monge matrix.*

(Permuted) Monge matrices possess many nice properties. For example, for the linear assignment problem with a Monge cost matrix, the identity permutation, $\epsilon(i) = i$ for $i = 1, ..., n$, is a simple optimal solution. We refer to [8], [25] and [24] for more applications of Monge matrices in combinatorial optimization.

We will show that our problem can be reduced to a special linear assignment problem with a product matrix as its cost matrix, therefore, it can be solved in $O(n \log n)$ time.

Define $x_{ij} = 1$ if job j is assigned to position i and $x_{ij} = 0$ otherwise. Our sequencing problem then reduces to the following linear assignment problem:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} \xi_i \eta_j x_{ij}$$

s.t.

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, 2, \cdots, n$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, 2, \cdots, n.$$

$$x_{ij} \in \{0,1\}, i,j = 1,2,\cdots,n.$$

By the Hardy-Littlewood-Polya inequality/principle [37], the objective function is minimized by the following algorithm:

Algorithm 1 Step 1. *Sort* $\{\xi_i\}$ *in decreasing order and* $\{\eta_j\}$ *in increasing order. For simplicity, assume they are already sorted in this way, i.e.,* $\xi_1 \geq \xi_2 \geq \cdots \geq \xi_n$, $\eta_1 \leq \eta_2 \leq \cdots \leq \eta_n$.

Step 2. *Let* $x_{ii} = 1$ *for* $i = 1,2,\cdots,n$. *Let* $x_{ij} = 0$ *for* $i \neq j$. *In other words, assign the* i*-th job to the* i*-th position.*

Therefore, we have the following result:

Lemma 2 *The optimal sequence, denoted by* π^*, *can be obtained by solving a special linear assignment problem requiring* $O(n \log n)$ *time.*

In the next subsections we present various applications of our general approach to solve a large set of important scheduling problems with variable processing times.

2.2 Applications to Solve Scheduling Problems

Let $C_{[k]}$ be the completion time of the kth job in the processing sequence whose processing time varies according to (1.3). The following crucial equality is given in [90].

Lemma 3 *Assume that the jobs are processed from time zero and there is no idle time between them, then*

$$C_{[k]} = \sum_{j=1}^{k} (j^c \prod_{i=j+1}^{k} (1 + bi^c)) a_{[j]} \tag{2.2}$$

where $\prod_{i=k+1}^{k}(1 + bi^c) = 1$ *by definition.*

Define

$$f_{jk} = j^c \prod_{i=j+1}^{k} (1 + bi^c) \tag{2.3}$$

for $j = 1,\cdots,n$ and $k = j+1,\cdots,n$, and $f_{jk} = 0$ for other values of j and k. Then equation (2.2) above can be rewritten as

$$C_{[k]} = \sum_{j=1}^{k} f_{jk} a_{[j]}. \tag{2.4}$$

2.2.1 Minimizing Earliness/Tardiness with Due Date Assignment

In this subsection, we show how our unified method can be used to solve a large set of scheduling problems involving due date assignment decisions.

Since the solution of due date assignment problems also includes the determination of the due dates, a schedule for these problems is defined by a job sequence $\pi = ([1], [2], ..., [n])$ and a due date assignment vector $\mathbf{d} = (d_1, d_2, ..., d_n)$, and our objective is to determine the schedule which minimizes various objective functions defined in the subsections below. Therefore, to also show their dependence on the due dates, we re-define $g(\pi)$ from (2.1) as $g(\pi, \mathbf{d})$. Cases will be considered by providing special forms of $g(\pi, \mathbf{d})$.

Our objective is to find a job sequence $\pi^* = ([1], [2], ..., [n])$ and a set of due dates $\mathbf{d}^* = (d_1^*, d_2^*, ..., d_n^*)$ to minimize a cost function that includes the costs of earliness, tardiness, due date assignment and makespan. For the *CON*, *SLK* and *DIF* due date assignment methods, it is given by

$$g(\pi, \mathbf{d}) = \alpha \sum_{j=1}^{n} E_j + \beta \sum_{j=1}^{n} T_j + \gamma \sum_{j=1}^{n} d_j + \delta C_{\max} \tag{2.5}$$

where for job j, C_j is the completion time; $E_j = [d_j - C_j]^+$ is the earliness; $T_j = [C_j - d_j]^+$ is the tardiness, (where $[x]^+ \stackrel{\text{def}}{=} \max(0, x)$). $C_{\max} = \max_{j=1,...,n} C_j$ is the maximum completion time (makespan) and α, β, γ and δ are nonnegative parameters representing the cost of one unit of earliness, tardiness, due date and operation time, respectively.

In the case of the *CONW* method, the cost function to minimize is given by

$$g(\pi, \mathbf{d}) = \alpha \sum_{j=1}^{n} E_j + \beta \sum_{j=1}^{n} T_j + n\left(\gamma_1 \underline{d} + \gamma_2 D\right) + \delta C_{\max}, \tag{2.6}$$

where earliness is defined by $E_j = \left[\underline{d}_j - C_j\right]^+$, tardiness is defined by $T_j = \left[C_j - \overline{d}_j\right]^+$ and the term $\gamma \sum_{j=1}^{n} d_j$ in eq. (2.5), is replaced by the term $\gamma_1 nd + \gamma_2 nD$.

It is easy to see that the jobs are processed from time zero and there is no idle time between them, i.e., the condition of Lemma 3 is satisfied for all the due date assignment methods. Therefore, (2.4) holds in all cases. Next we show that, under an optimal due date assignment strategy, the scheduling cost can be reduced to the format of (2.1) for all of them.

Due date assignment problems have been extensively studied in the literature with various forms of objectives. Among them, however, there are few results with learning effect or time-dependent processing time. Below are these results as far as we know:

We notice they all use CON due date assignment method. In this paper, we both improve the time complexity of and extend the first two, and extend the third one. Furthermore, for the first time we present results with other due date assignment methods, i.e., CONW, DIF and SLK.

Problem	Complexity	Ref.
$1\left\|p_{[j]}=a_{[j]}j^c, CON\right\| \alpha\sum E_j+\beta\sum T_j+\theta\sum C_j$	$O(n^3)$	[5]
$1\left\|p_{[j]}=a_{[j]}j^c, CON\right\| \alpha\sum E_j+\beta\sum T_j+\gamma\sum d_j$	$O(n^3)$	[61]
$1\left\|p_{[j]}=a_{[j]}+bS_{[j]}, CON\right\| \alpha\sum E_j+\beta\sum T_j+\gamma\sum d_j$	$O(n\log n)$	[19]

Table 2.1: Summary of previous results on due date assignment and learning effect/time-dependent processing time.

Preliminary results for the *CON* due date assignment method

The following result is given by Panwalkar *et al.* [67] for the *CON* due date assignment method.

Lemma 4 *For the CON due date assignment method, there exists an optimal due date equal to* $C_{[l^*]}$*, where*

$$l^*=\max\left(\left\lceil\frac{n(\beta-\gamma)}{\alpha+\beta}\right\rceil,0\right),\ \textit{and } C_{[0]}=0 \textit{ by definition.} \tag{2.7}$$

Note that the value of l^*, given by eq.(2.7), is independent of the job processing times and the job sequence. Therefore, it is optimal for *any* job sequence and processing times, and substituting (2.4), we obtain:

$$d_j^*=d^*=C_{[l^*]}=\sum_{i=1}^{l^*}f_{il^*}a_{[i]}\ \text{ for } j=1,...,n; \tag{2.8}$$

$$E_{[j]}=\begin{cases}\sum_{i=j+1}^{l^*}p_{[i]}=C_{[l^*]}-C_{[j]}=C_{[l^*]}-\sum_{i=1}^{j}f_{ij}a_{[i]} & j=1,2,\cdots,l^*-1\\ 0 & j=l^*,\cdots,n\end{cases}$$

and

$$T_{[j]}=\begin{cases}0 & j=1,2,\cdots,l^*\\ C_{[j]}-C_{[l^*]}=\sum_{i=1}^{l^*}(f_{ij}-f_{il^*})a_{[i]}+\sum_{i=l^*+1}^{j}f_{ij}a_{[i]} & j=l^*+1,\cdots,n\end{cases}$$

Therefore, by changing the order of summation, the total earliness can be expressed as

$$\sum_{j=1}^{l^*}E_{[j]}=(l^*-1)C_{[l^*]}-\sum_{j=1}^{l^*}\sum_{i=1}^{j}f_{ij}a_{[i]}=\sum_{i=1}^{l^*}(l^*f_{il^*}-\sum_{j=i}^{l^*}f_{ij})a_{[i]} \tag{2.9}$$

and the total tardiness is

$$\begin{aligned}\sum_{j=l^*+1}^{n} T_{[j]} &= \sum_{j=l^*+1}^{n}\sum_{i=1}^{l^*}(f_{ij}-f_{il^*})a_{[i]} + \sum_{j=l^*+1}^{n}\sum_{i=l^*+1}^{j} f_{ij}a_{[i]} \\ &= \sum_{i=1}^{l^*}((l^*-n)f_{il^*} + \sum_{j=l^*+1}^{n} f_{ij})a_{[i]} + \sum_{i=l^*+1}^{n}\sum_{j=i}^{n} f_{ij}a_{[i]}. \end{aligned} \tag{2.10}$$

By substituting eqs. (2.8)-(2.10) into eq. (2.5), we obtain that our scheduling cost, under an optimal due date assignment strategy, denoted by $\mathbf{d}^*(\pi)$, for the *CON* due date assignment method becomes:

$$\begin{aligned} g(\pi,\mathbf{d}^*(\pi)) &= \sum_{i=1}^{l^*}\left(\alpha(l^*-1)f_{il^*} - \alpha\sum_{j=i}^{l^*} f_{ij} + \beta(l^*-1-n)f_{il^*} + \beta\sum_{j=l^*+1}^{n} f_{ij} + \gamma n f_{il^*} + \delta f_{in}\right)a_{[i]} \\ &\quad + \sum_{i=l^*+1}^{n}(\beta\sum_{j=i}^{n} f_{ij} + \delta f_{in})a_{[i]}. \end{aligned} \tag{2.11}$$

Note that if the upper bound of summation is less than the lower bound of summation, the summation is 0. It is easy to observe that the objective $g(\pi,\mathbf{d}^*(\pi))$ in eq. (2.11) is a special case of $g(\pi)$ in (2.1) with

$$\xi_i = \begin{cases} \alpha(l^*-1)f_{il^*} - \alpha\sum_{j=i}^{l^*} f_{ij} + \beta(l^*-1-n)f_{il^*} + \beta\sum_{j=l^*+1}^{n} f_{ij} + \gamma n f_{il^*} + \delta f_{in} & i=1,\cdots,l^* \\ \beta\sum_{j=i}^{n} f_{ij} + \delta f_{in} & i=l^*+1,\cdots,n \end{cases} \tag{2.12}$$

This equation becomes much simpler if there is only learning effect or only time-dependent processing time. In fact, if there is only learning effect, we have $b=0$, therefore f_{ij} is reduced to i^c. Thus the equation above becomes

$$\xi_i = \begin{cases} \alpha(i-1)i^c + (\gamma n+\delta)i^c & i=1,\cdots,l^* \\ \beta(n-i+1)i^c + \delta i^c & i=l^*+1,\cdots,n \end{cases} \tag{2.13}$$

From this simplified formula, we can show that learning effect does affect optimal scheduling. We now construct an example to demonstrate that the optimal sequence is different when the learning effect is in place. By Lemma 2, it suffices to show that the ξ_i order sequence is different.

In our example, there are only 3 jobs, i.e., $n=3$, $a_1 < a_2 < a_3$. $\alpha=1,\beta=2,\gamma=\delta=0$. From Lemma 4, we have $l^*=2$. If the learning factor $c=-2$, using the equation above, we have $\xi_1=0$, $\xi_2=1/4 > \xi_3=2/9$. Therefore, by Lemma 2, in the optimal sequence, job 2 should

precede job 3. However, if there is no learning effect, i.e., $c = 0$, we have $\xi_1 = 0$, $\xi_2 = 1 < \xi_3 = 2$. In this case, job 3 should precede job 2 in the optimal sequence.

On the other hand, if there is only time-dependent processing time, i.e., $c = 0$, then f_{ij} is reduced to $(1+b)^{j-i}$. Thus the expression above becomes

$$\xi_i = \begin{cases} (\alpha l^* + \beta(l^* - n) + \gamma n)(1+b)^{l^*-i} - \alpha \sum_{j=i}^{l^*}(1+b)^{j-i} & \\ +\beta \sum_{j=l^*+1}^{n}(1+b)^{j-i} + \delta(1+b)^{n-i} & i = 1, \cdots, l^* \\ \beta \sum_{j=i}^{n}(1+b)^{j-i} + \delta(1+b)^{n-i} & i = l^*+1, \cdots, n \end{cases} \tag{2.14}$$

Similar to the case of learning effect only, we can also construct an example to show that time-dependent processing time does affect the optimal scheduling.

Preliminary results for the SLK due date assignment method

For fixed processing times, Adamopoulos and Pappis [1] showed that the *CON* and the *S-LK* methods have similar properties and presented the following result for the *SLK* due date assignment method.

Lemma 5 *For the SLK due date assignment method, there exists an optimal slack allowance, s^*, equal to $C_{[l^*-1]}$, where l^* is given by eq. (2.7) if $l^* > 0$; $s^* = 0$ otherwise.*

As a result, if we let $h = l^* - 1$, $j' = j - 1$, for $j = 1, \cdots, n$ and substitute (2.4), we obtain the following for any $\pi = ([1], [2], \cdots, [n])$:

$$s^* = C_{[h]} = \sum_{i=1}^{h} f_{ih} a_{[i]}; \tag{2.15}$$

$$d^*_{[j]} = p_{[j]} + s^* = p_{[j]} + \sum_{i=1}^{h} f_{ih} a_{[i]} \quad \text{for} \quad j = 1, ..., n; \tag{2.16}$$

$$\gamma \sum_{j=1}^{n} d^*_{[j]} = \gamma C_{[h]} + n\gamma \sum_{i=1}^{h} f_{ih} a_{[i]} = \gamma \sum_{i=1}^{n} f_{in} a_{[i]} + n\gamma \sum_{i=1}^{h} f_{ih} a_{[i]} \tag{2.17}$$

$$E_{[j]} = \begin{cases} C_{[h]} - C_{[j']} = C_{[h]} - \sum_{i=1}^{j'} f_{ij'} a_{[i]} & j = 1, 2, \cdots, h \\ 0 & j = h+1, \cdots, n \end{cases}$$

and

$$T_{[j]} = \begin{cases} 0 & j = 1, 2, \cdots, h+1 \\ C_{[j']} - C_{[h]} = \sum_{i=1}^{h}(f_{ij'} - f_{ih}) a_{[i]} + \sum_{i=h+1}^{j'} f_{ij'} a_{[i]} & j = h+2 \cdots, n \end{cases}$$

Therefore, by changing the order of summation and substituting (2.3) for f_{hh}, the total earliness is

$$\sum_{j=1}^{h} E_{[j]} = hC_{[h]} - \sum_{j'=0}^{h-1}\sum_{i=1}^{j'} f_{ij'}a_{[i]} = \sum_{i=1}^{h-1}(hf_{ih} - \sum_{j=i}^{h-1} f_{ij})a_{[i]} + h^{c+1}a_{[h]}, \tag{2.18}$$

where for the last term we used $hf_{hh}a_{[h]} = hh^c\prod_{i=h+1}^{h}(1+bi^c)a_{[h]} = h^{c+1}a_{[h]}$. Similarly, the total tardiness is

$$\begin{aligned}\sum_{j=h+2}^{n} T_{[j]} &= \sum_{j'=h+1}^{n-1}(\sum_{i=1}^{h}(f_{ij'} - f_{ih})a_{[i]} + \sum_{i=h+1}^{j'} f_{ij'}a_{[i]}) \\ &= \sum_{i=1}^{h}(\sum_{j'=h+1}^{n-1} f_{ij'} - (n-h-1)f_{ih})a_{[i]} + \sum_{i=h+1}^{n-1}(\sum_{j'=i}^{n-1} f_{ij'})a_{[i]}\end{aligned} \tag{2.19}$$

By substituting eqs.(2.17)-(2.19) into eq.(2.5), we obtain that our scheduling cost, under an optimal due date assignment strategy, denoted by $\mathbf{d}^*(\pi)$, for the *SLK* due date assignment method becomes:

$$\begin{aligned}g(\pi,\mathbf{d}^*(\pi)) &= \sum_{i=1}^{h-1}(\alpha h f_{ih} - \alpha\sum_{j=i}^{h-1} f_{ij} + \beta\sum_{j=h+1}^{n-1} f_{ij} - \beta(n-h-1)f_{ih} + \gamma n f_{ih} + (\gamma+\delta)f_{in})a_{[i]} \\ &\quad +(\alpha h^{c+1} + \beta(\sum_{j'=h+1}^{n-1} f_{hj'} - (n-h-1)h^c) + (\gamma+\delta)f_{hn})a_{[h]} \\ &\quad + \sum_{i=h+1}^{n-1}(\beta\sum_{j'=i}^{n-1} f_{ij'} + (\gamma+\delta)f_{in})a_{[i]} + (\gamma+\delta)n^c a_{[n]}.\end{aligned} \tag{2.20}$$

It is easy to observe that the objective $g(\pi,\mathbf{d}^*(\pi))$ in eq. (2.20) is a special case of $g(\pi)$ in (2.1) if we define the positional penalties by

$$\xi_i = \begin{cases} \alpha h f_{ih} - \alpha\sum_{j=i}^{h-1} f_{ij} + \beta\sum_{j=h+1}^{n-1} f_{ij} - \beta(n-h-1)f_{ih} & \\ +\gamma n f_{ih} + (\gamma+\delta)f_{in} & i = 1,\cdots,h-1 \\ \alpha h^{c+1} + \beta(\sum_{j=h+1}^{n-1} f_{hj} - (n-h-1)h^c) + (\gamma+\delta)f_{hn} & i = h \\ \beta\sum_{j=i}^{n-1} f_{ij} + (\gamma+\delta)f_{in} & i = h+1,\cdots,n-1 \\ (\gamma+\delta)n^c & i = n \end{cases} \tag{2.21}$$

This equation becomes much simpler if there is only learning effect or only time-dependent processing time. In fact, if there is only learning effect, we have $b = 0$, therefore f_{ij} is reduced to i^c. Thus the equation above becomes

$$\xi_i = \begin{cases} (\alpha i + \gamma n + \gamma + \delta) i^c & i = 1, \cdots, h-1 \\ \alpha h^{c+1} + (\gamma + \delta) h^c & i = h \\ \beta(n-i) i^c + (\gamma + \delta) i^c & i = h+1, \cdots, n-1 \\ (\gamma + \delta) n^c & i = n \end{cases} \tag{2.22}$$

Preliminary results for the *DIF* due date assignment method

The DIF due date assignment method to minimize earliness, tardiness and due date assignment costs was studied by Seidmann *et al.* [78] and they presented the following lemma which defines the optimal due date assignment strategy for a given π and non-variable processing times.

Lemma 6 *For a given π and fixed processing times, the optimal due date assignment strategy for the DIF due date assignment method is defined as follows: if $\gamma \geq \beta$ then set $d^*_{[j]} = 0$; otherwise, set $d^*_{[j]} = C_{[j]}$ for $j = 1, ..., n$.*

From Lemma 6, we can conclude that $E_{[j]} = 0$ for $j = 1, ..., n$ in an optimal solution for our problem. Therefore, with an optimal due date assignment strategy as a function of π, after substituting (2.4), eq. (2.5) becomes

$$g(\pi, \mathbf{d}^*(\pi)) = \begin{cases} \beta \sum_{k=1}^{n} C_{[k]} + \delta C_{[n]} = \beta \sum_{k=1}^{n} \sum_{j=1}^{k} f_{jk} a_{[j]} + \delta C_{[n]} & \\ \qquad = \beta \sum_{j=1}^{n} \sum_{k=j}^{n} f_{jk} a_{[j]} + \delta C_{[n]} & \text{if } \gamma \geq \beta \\ \gamma \sum_{k=1}^{n} C_{[k]} + \delta C_{[n]} = \gamma \sum_{k=1}^{n} \sum_{j=1}^{k} f_{jk} a_{[j]} + \delta C_{[n]} & \\ \qquad = \gamma \sum_{j=1}^{n} \sum_{k=j}^{n} f_{jk} a_{[j]} + \delta C_{[n]} & \text{if } \gamma < \beta. \end{cases}$$

Using (2.4) for $C_{[n]}$, this can be further written as

$$g(\pi, \mathbf{d}^*(\pi)) = \sum_{j=1}^{n} (\epsilon \sum_{k=j}^{n} f_{jk} + \delta f_{jn}) a_{[j]}, \tag{2.23}$$

where $\epsilon = \min(\beta, \gamma)$.

Again, as for the previous two due date assignment methods, it is easy to observe that the objective $g(\pi, \mathbf{d}^*(\pi))$ in eq. (2.23) is a special case of $g(\pi)$ in (2.1) with

$$\xi_i = \epsilon \sum_{k=i}^{n} f_{ik} + \delta f_{in} \text{ for } i = 1, ..., n. \tag{2.24}$$

.

Preliminary results for the *CONW* due date assignment method

For fixed processing times, Liman *et al.* [57] presented the following result.

Lemma 7 *[57] Calculate*

$$l_1^* = \min\left(\max\left(\left\lfloor \frac{n(\gamma_2 - \gamma_1)}{\alpha} + 1 \right\rfloor, 0\right), n\right) \text{ and } l_2^* = \max\left(\left\lfloor \frac{n(\beta - \gamma_2)}{\beta} + 1 \right\rfloor, 0\right), \tag{2.25}$$

where γ_1, γ_2 are from (2.6). If $l_1^* < l_2^*$, then there exists an optimal $\underline{d}^*$ equal to $C_{[l_1^*]}$ and optimal $\overline{d}^*$ equal to $C_{[l_2^*]}$; if $l_1^* \geq l_2^*$, then there exists an optimal window of zero length, so the case reduces to the CON *case with $\underline{d}^* = \overline{d}^*$ equal to $C_{[l^*]}$, where l^* is calculated by eq. (2.7) using $\gamma = \gamma_1$, and $C_{[0]} = 0$ by definition.*

We note that the calculation of the indexes l_1^* and l_2^* in the above lemma depends only on the unit-cost parameters and n, and it does *not* depend on the job processing times. This means that we can determine, independently from the job processing times, which one of the above two cases an instance falls into. According to Lemma 7, if $l_1^* \geq l_2^*$, then the optimal due-window strategy is identical to the optimal *CON* strategy and therefore eqs. (2.8)-(2.12) hold for *CONW* with $\gamma = \gamma_1$. Otherwise, i.e., if $l_1^* < l_2^*$, then the following holds for any π after substituting (2.4):

$$\underline{d}^* = C_{[l_1^*]} = \sum_{i=1}^{l_1^*} f_{il_1^*} a_{[i]}, \quad \overline{d}^* = C_{[l_2^*]} = \sum_{i=1}^{l_2^*} f_{il_2^*} a_{[i]}; \quad D = \overline{d}^* - \underline{d}^* = \sum_{i=1}^{l_2^*} f_{il_2^*} a_{[i]} - \sum_{i=1}^{l_1^*} f_{il_1^*} a_{[i]} \tag{2.26}$$

$$E_{[j]} = \begin{cases} C_{[l_1^*]} - C_{[j]} = C_{[l_1^*]} - \sum_{i=1}^{j} f_{ij} a_{[i]} & \text{for} \quad j = 1, ..., l_1^* - 1 \\ 0 & \text{for} \quad j = l_1^*, ..., n \end{cases}; \tag{2.27}$$

$$T_{[j]} = \begin{cases} 0 & \text{for} \quad j = 1, ..., l_2^* \\ C_{[j]} - C_{[l_2^*]} = \sum_{i=1}^{l_2^*} (f_{ij} - f_{il_2^*}) a_{[i]} + \sum_{i=l_2^*+1}^{j} f_{ij} a_{[i]} & \text{for} \quad j = l_2^* + 1, ..., n. \end{cases} \tag{2.28}$$

Therefore, after substituting (2.4) and changing the order of summation, the total earliness is

$$\sum_{j=1}^{l_1^*} E_{[j]} = \sum_{i=1}^{l_1^*} (l_1^* f_{il_1^*} - \sum_{j=i}^{l_1^*} f_{ij}) a_{[i]} \tag{2.29}$$

Similarly, the total tardiness is

$$\begin{aligned} \sum_{j=l_2^*+1}^{n} T_{[j]} &= \sum_{j=l_2^*+1}^{n} \sum_{i=1}^{l_2^*} (f_{ij} - f_{il_2^*}) a_{[i]} + \sum_{j=l_2^*+1}^{n} \sum_{i=l_2^*+1}^{j} f_{ij} a_{[i]} \\ &= \sum_{i=1}^{l_2^*} ((l_2^* - n) f_{il_2^*} + \sum_{j=l_2^*+1}^{n} f_{ij}) a_{[i]} + \sum_{i=l_2^*+1}^{n} (\sum_{j=i}^{n} f_{ij}) a_{[i]} \end{aligned} \tag{2.30}$$

By substituting eqs. (2.26)-(2.30) into eq.(2.6), we obtain that our scheduling cost, under an optimal due date assignment strategy, denoted by $\mathbf{d}^*(\pi)$, for the *CONW* due date assignment method becomes:

$$\begin{aligned} g(\pi, \mathbf{d}^*(\pi) = &\sum_{i=1}^{l_1^*} \left(\alpha(l^* f_{il_1^*} - \sum_{j=i}^{l_1^*} f_{ij}) + \beta((l_2^* - n) f_{il_2^*} + \sum_{j=l_2^*+1}^{n} f_{ij}) + n(\gamma_1 - \gamma_2) f_{il_1^*} + n\gamma_2 f_{il_2^*} + \delta f_{in} \right) a_{[i]} \\ &+ \sum_{i=l_1^*+1}^{l_2^*} \left(n\gamma_2 f_{il_2^*} + \beta((l_2^* - n) f_{il_2^*} + \sum_{j=l_2^*+1}^{n} f_{ij}) + \delta f_{in} \right) a_{[i]} + \sum_{i=l_2^*+1}^{n} \left(\beta \sum_{j=i}^{n} f_{ij} + \delta f_{in} \right) a_{[i]}. \end{aligned} \tag{2.31}$$

Note that $g(\pi, \mathbf{d}^*(\pi))$ in eq. (2.31) is a special case of $g(\pi)$ in (2.1) if the positional penalties are defined by

$$\xi_i = \begin{cases} \alpha(l^* f_{il_1^*} - \sum_{j=i}^{l_1^*} f_{ij}) + \beta((l_2^* - n) f_{il_2^*} + \sum_{j=l_2^*+1}^{n} f_{ij}) & \\ +n(\gamma_1 - \gamma_2) f_{il_1^*} + n\gamma_2 f_{il_2^*} + \delta f_{in} & \text{for} \quad i = 1, ..., l_1^* \\ n\gamma_2 f_{il_2^*} + \beta((l_2^* - n) f_{il_2^*} + \sum_{j=l_2^*+1}^{n} f_{ij}) + \delta f_{in} & \text{for} \quad i = l_1^* + 1, ..., l_2^* \\ \beta \sum_{j=i}^{n} f_{ij} + \delta f_{in} & \text{for} \quad i = l_2^* + 1, ..., n. \end{cases} \tag{2.32}$$

This equation becomes much simpler if there is only learning effect or only time-dependent processing time. In fact, if there is only learning effect, we have $b = 0$, therefore f_{ij} is reduced

to i^c. Thus the equation above becomes

$$\xi_i = \begin{cases} \alpha(l^* - l_1^* + i - 1)i^c + n(\gamma_1 - \gamma_2)i^c + n\gamma_2 i^c + \delta i^c & \text{for} \quad i = 1, ..., l_1^* \\ n\gamma_2 i^c + \delta i^c & \text{for} \quad i = l_1^* + 1, ..., l_2^* \\ \beta(n - i + 1)i^c + \delta i^c & \text{for} \quad i = l_2^* + 1, ..., n. \end{cases} \tag{2.33}$$

Application of the Unified Optimization Algorithm According to the analysis above, the objective function (2.5) or (2.6) can be written in the format of (2.1) for all four due date assignment methods, and thus we can present the following optimization algorithm to solve the problem for the earliness-tardiness objective:

Algorithm 2 *A unified optimization algorithm for solving the* $1\left|p_{[j]} = (a_{[j]} + bS_{[j]})j^c, CONW\right| \alpha\sum E_j + \beta\sum T_j + n(\gamma_1 d + \gamma_2 D) + \delta C_{\max}$ *and the* $1\left|p_{[j]} = (a_{[j]} + bS_{[j]})j^c, X\right| \alpha\sum E_j + \beta\sum T_j + \gamma\sum d_j + \delta C_{\max}$ *problems for* $X \in \{CON, SLK, DIF\}$.

Step 1. (Apply this step only for the CON, CONW and the SLK methods) Calculate l^* by eq. (2.7) for the *CON* and the *SLK* methods and l_1^* and l_2^* by eq. (2.25) for the *CONW* method.

Step 2. Apply Algorithm 1 where ξ_i is calculated by eq. (2.12) for the *CON* method and the *CONW* method if $l_1^* \geq l_2^*$, by eq. (2.32) for the *CONW* method if $l_1^* < l_2^*$, by eq. (2.21) for the *SLK* method and by eq. (2.24) for the *DIF* method.

Step 3. For the *CON* method, assign the due date according to eqs. (2.7)-(2.8). For the SLK method, assign the due dates according to eqs. (2.15)-(2.16). For the DIF method, assign the due dates according to Lemma 6. For the *CONW* method, if $l_1^* \geq l_2^*$ assign $\underline{d}^* = \overline{d}^*$ where $\underline{d}^*$ is calculated by eqs. (2.7)-(2.8) with $\gamma = \gamma_1$; If $l_1^* < l_2^*$, assign $\underline{d}^*$ and $\overline{d}^*$ according to eq. (2.26).

We have the following result for this algorithm:

Theorem 1 *Algorithm 2 solves the earliness-tardiness scheduling problems* $1|p_{[j]} = (a_{[j]}+bS_{[j]})j^c, CONW|\alpha\sum E_j+\beta\sum T_j+n(\gamma_1 d+\gamma_2 D)+\delta C_{\max}$ *and* $1\left|p_{[j]} = (a_{[j]} + bS_{[j]})j^c, X\right| \alpha\sum E_j+\beta\sum T_j+\gamma\sum d_j + \delta C_{\max}$ *for* $X \in \{CON, SLK, DIF\}$ *in* $O(n\log n)$ *time.*

Proof. The correctness of the algorithm follows from Lemmas 2-4 and the preceding analysis. Step 1 takes constant time and Step 3 requires $O(n)$ time; Step 2 requires $O(n\log n)$ time, and the calculation of ξ_i for $i = 1, 2, \cdots, n$. We will show that this can be done in linear time.

Examine (2.12), (2.24), (2.21) and (2.32), we show that all the positional penalties can be computed in linear time. The calculation of ξ_i includes the calculation of three types of expressions: f_{im} ($m = l^*$ or n in (2.12)), $\sum_{j=m}^{l} f_{ij}$ ($m = l^* + 1, l = n$ in (2.12)) and $\sum_{j=i}^{l} f_{ij}$ ($l = l^*$ or

n in (2.12)) for i of certain range ($i = 1, \cdots, l^*$ or $i = l^* + 1 \cdots, n$ in (2.12)), where m and l are some *fixed* values depending on the due date assignment method used.

Let $e_{ij} = \prod_{k=i+1}^{j}(1 + bk^c)$. Then $f_{ij} = i^c e_{ij}$ by (2.3). Thus the calculation of f_{ij} and $\sum_j f_{ij}$ is reduced to the calculation of e_{ij} and $\sum_j e_{ij}$. We can do this with the following backward recursions:

$e_{mm} = 1$ and $\{e_{im} | 1 \le i \le m\}$ can be calculated by $e_{i-1,m} = (1 + bi^c)e_{im}$ for $i = m, m-1, \cdots, 1$ in linear time.

$\sum_{j=m}^{l} e_{mj} = \sum_{j=m}^{l} \prod_{k=m+1}^{j}(1+bk^c)$ can clearly be obtained in linear time, and using $\sum_{j=m}^{l} e_{i-1,j} = (1 + bi^c) \sum_{j=m}^{l} e_{ij}$ for $i = m, m-1, \cdots, 1$, each of $\{\sum_{j=m}^{l} e_{ij} | 1 \le i \le m\}$ can be calculated in $O(1)$ time from the previous value.

$\sum_{j=l}^{l} e_{lj} = 1$ and using $\sum_{j=i-1}^{l} e_{i-1,j} = 1 + (1 + bi^c) \sum_{j=i}^{l} e_{ij}$ for $i = l, l-1, \cdots, 1$, each of $\{\sum_{j=i}^{l} e_{ij} | 1 \le i \le l\}$ can be calculated in $O(1)$ time from the previous entry. To demonstrate, we show the first 2 rounds of these iterations below:

$$\sum_{j=l-1}^{l} e_{l-1,j} = 1 + (1 + bl^c) \sum_{j=l}^{l} e_{lj} = 1 + (1 + bl^c),$$

$$\sum_{j=l-2}^{l} e_{l-2,j} = 1 + (1 + b(l-1)^c) \sum_{j=l-1}^{l} e_{l-1,j} = 1 + (1 + b(l-1)^c)(1 + (1 + bl^c)).$$

Thus the overall computational complexity of the algorithm is $O(n \log n)$ indeed. Similar arguments show that the calculations in (2.24), (2.21) and (2.32) can also be implemented in linear time. ■

As special cases, we have the following results, when there is only learning effect, or only time-dependent processing times, respectively:

Corollary 1 *The earliness-tardiness scheduling problems with learning effects* $1|p_{[j]} = a_{[j]}j^c$, $CONW|\alpha \sum E_j + \beta \sum T_j + n(\gamma_1 d + \gamma_2 D) + \delta C_{\max}$ *and* $1\left|p_{[j]} = a_{[j]}j^c, X\right| \alpha \sum E_j + \beta \sum T_j + \gamma \sum d_j + \delta C_{\max}$ *for* $X \in \{CON, SLK, DIF\}$ *are solvable in* $O(n \log n)$ *time.*

Corollary 2 *The earliness-tardiness scheduling problems with time-dependent processing times* $1|p_{[j]} = a_{[j]} + bS_{[j]}$, $CONW|\alpha \sum E_j + \beta \sum T_j + n(\gamma_1 d + \gamma_2 D) + \delta C_{\max}$ *and* $1|p_{[j]} = a_{[j]} + bS_{[j]}, X|$ $\alpha \sum E_j + \beta \sum T_j + \gamma \sum d_j + \delta C_{\max}$ *for* $X \in \{CON, SLK, DIF\}$ *are solvable in* $O(n \log n)$ *time.*

It is easy to see that, if instead of makespan, we consider the total completion time in the objective of Theorem 1, i.e., we replace $\delta C_{\max}$ with $\theta \sum C_j$, the same argument holds. The only difference is that in the definition of ξ_i, we use $\theta \sum_{k=i}^{n} f_{ik}$ instead of δf_{in}. Therefore we also have the following result:

Theorem 2 *The earliness-tardiness scheduling problems* $1|p_{[j]} = (a_{[j]}+bS_{[j]})j^c, CONW|\alpha\sum E_j+\beta\sum T_j+n(\gamma_1 d+\gamma_2 D)+\theta\sum C_j$ *and* $1\left|p_{[j]} = (a_{[j]}+bS_{[j]})j^c, X\right|\alpha\sum E_j+\beta\sum T_j+\gamma\sum d_j+\theta\sum C_j$ *for* $X \in \{CON, SLK, DIF\}$ *are solvable in* $O(n\log n)$ *time.*

Again, as special cases, we have the following results, when there is only learning effect, or only time-dependent processing times, respectively:

Corollary 3 *The earliness-tardiness scheduling problems with learning effects* $1|p_{[j]} = a_{[j]}j^c$, $CONW|\alpha\sum E_j+\beta\sum T_j+n(\gamma_1 d+\gamma_2 D)+\theta\sum C_j$ *and* $1\left|p_{[j]} = a_{[j]}j^c, X\right|\alpha\sum E_j+\beta\sum T_j+\gamma\sum d_j+\theta\sum C_j$ *for* $X \in \{CON, SLK, DIF\}$ *are solvable in* $O(n\log n)$ *time.*

Corollary 4 *The earliness-tardiness scheduling problems with time-dependent processing times* $1|p_{[j]} = a_{[j]}+bS_{[j]}$, $CONW|\alpha\sum E_j+\beta\sum T_j+n(\gamma_1 d+\gamma_2 D)+\theta\sum C_j$ *and* $1|p_{[j]} = a_{[j]}+bS_{[j]}, X|\alpha\sum E_j+\beta\sum T_j+\gamma\sum d_j+\theta\sum C_j$ *for* $X \in \{CON, SLK, DIF\}$ *are solvable in* $O(n\log n)$ *time.*

Our results both extend and improve Biskup's [5] $O(n^3)$ time optimization algorithm to solve the $1\left|p_{[j]} = a_{[j]}j^c, CON\right|\ \alpha\sum E_j+\beta\sum T_j+\theta\sum C_j$ problem, and Mosheiov's [61] $O(n^3)$ time optimization algorithm to solve the $1|p_{[j]} = a_{[j]}+bS_{[j]}, CON|\alpha\sum E_j+\beta\sum T_j+\gamma\sum d_j$ problem.

2.2.2 Scheduling Problems without Due Dates

We show in this subsection that a large collection of other classical scheduling problems, which do not involve due dates, can also be solved by applying Algorithm 2.

It is easy to see that the jobs are processed from time zero and there is no idle time between them, i.e., the condition of Lemma 3 is satisfied for all the objectives without due dates. Therefore, (2.4) holds in all cases.

Below is a list of known results about scheduling problems for learning effect or time dependent processing time without due date assignment consideration:

Makespan

Let us first consider the *makespan* minimization problem with variable processing times, denoted by $1|p_{[j]} = (a_{[j]}+bS_{[j]})j^c|C_{\max}$.

When $k = n$, (2.4) becomes:

$$C_{max} = C_{[n]} = \sum_{i=1}^{n} f_{in}a_{[i]}.$$

Thus we can write the objective using "positional" penalties to minimize

Problem	Complexity	Ref.
$1\left\|p_{[j]}=a_{[j]}j^c\right\|C_{\max}$	$O(n\log n)$	[61]
$1\left\|p_{[j]}=a_{[j]}+bS_{[j]}\right\|C_{\max}$	$O(n\log n)$	[35]
$1\left\|p_{[j]}=a_{[j]}j^c\right\|\sum C_j$	$O(n\log n)$	[5]
$1\left\|p_{[j]}=a_{[j]}+bS_{[j]}\right\|\sum C_j$	$O(n\log n)$	[66]
$1\left\|p_{[j]}=(a_{[j]}+bS_{[j]})j^c\right\|C_{\max}$	$O(n\log n)$	[90]
$1\left\|p_{[j]}=(a_{[j]}+bS_{[j]})j^c\right\|\sum C_j$	$O(n\log n)$	[90]
$1\left\|p_{[j]}=a_{[j]}j^c\right\|\delta_1\sum\sum\|C_i-C_j\|+\delta_2\sum C_j$	$O(n^3)$	[61]

Table 2.2: Summary of previous results on learning effect/time-dependent processing time without due date assignment.

$$g(\pi)=\sum_{i=1}^{n}f_{in}a_{[i]}, \tag{2.34}$$

which is clearly in the format of (2.1) with $\xi_i=f_{in}$ for $i=1,...,n$. Therefore, the problem can be solved in $O(n\log n)$ time by applying Algorithm 1. Thus we have the following result, which extends previous results by Mosheiov [61] for the $1\left|p_{[j]}=a_{[j]}j^c\right|C_{\max}$ problem, and by Gupta and Gupta [35] for the $1\left|p_{[j]}=a_{[j]}+bS_{[j]}\right|C_{\max}$ problem.

Theorem 3 *The* $1\left|p_{[j]}=(a_{[j]}+bS_{[j]})j^c\right|C_{\max}$ *problem is solvable in* $O(n\log n)$ *time.*

Total completion time

Now consider $1\left|p_{[j]}=(a_{[j]}+bS_{[j]})j^c\right|\sum C_j$. From (2.4), we have

$$\sum_{k=1}^{n}C_{[k]}=\sum_{k=1}^{n}\sum_{j=1}^{k}f_{jk}a_{[j]}=\sum_{j=1}^{n}\sum_{k=j}^{n}f_{jk}a_{[j]}. \tag{2.35}$$

For any scheduling sequence $\pi=([1],[2],...,[n])$, the objective value can be represented by

$$g(\pi)=\sum_{k=1}^{n}\sum_{j=1}^{k}f_{jk}a_{[j]}=\sum_{j=1}^{n}\sum_{k=j}^{n}f_{jk}a_{[j]}. \tag{2.36}$$

This objective in eq. (2.36) is again a special case of the one in eq.(2.1) with $\xi_j=\sum_{k=j}^{n}f_{jk}$ for $j=1,...,n$. Therefore we have the following result, which extends the results of Biskup [5] for the $1\left|p_{[j]}=a_{[j]}j^c\right|\sum C_j$ problem, and those of Ng. et al. [66] for the $1\left|p_{[j]}=a_{[j]}+bS_{[j]}\right|\sum C_j$ problem.

Theorem 4 *The* $1\left|p_{[j]}=(a_{[j]}+bS_{[j]})j^c\right|\sum C_j$ *problem is solvable in* $O(n\log n)$ *time.*

We note that the above solution does not extend to the weighted completion time problem, whose complexity remains unknown, since this objective function cannot be written using the positional-penalty format.

Notice that our algorithm for both makespan and total completion time is equivalent to the SPT algorithm in [90], since ξ_i is monotonically decreasing in i for both cases.

Sum and variation of completion times

Let us consider next the $1\left|p_{[j]}=(a_{[j]}+bS_{[j]})j^c\right|\delta_1\sum_{k=1}^{n}\sum_{l=k+1}^{n}(C_{[l]}-C_{[k]})+\delta_2\sum_{j=1}^{n}C_{[j]}$ problem. Substituting (2.4), we obtain

$$\begin{aligned}
\sum_{k=1}^{n}\sum_{l=k+1}^{n}(C_{[l]}-C_{[k]}) &= \sum_{k=1}^{n}\sum_{l=k+1}^{n}C_{[l]}-\sum_{k=1}^{n}\sum_{l=k+1}^{n}C_{[k]}\\
&= \sum_{l=1}^{n}\sum_{k=1}^{l-1}C_{[l]}-\sum_{k=1}^{n}(n-k)C_{[k]}\\
&= \sum_{l=1}^{n}(l-1)\sum_{j=1}^{l}f_{jl}a_{[j]}-\sum_{k=1}^{n}(n-k)\sum_{j=1}^{k}f_{jk}a_{[j]}\\
&= \sum_{j=1}^{n}\sum_{l=j}^{n}(l-1)f_{jl}a_{[j]}-\sum_{j=1}^{n}\sum_{k=j}^{n}(n-k)f_{jk}a_{[j]}\\
&= \sum_{j=1}^{n}\sum_{k=j}^{n}(2k-n-1)f_{jk}a_{[j]}
\end{aligned}$$

Note that since f_{jk} is increasing in k, each term is always non-negative. Adding (2.35), we can rewrite our objective as follows:

$$g(\pi)=\sum_{j=1}^{n}\left(\delta_1\sum_{k=j}^{n}(2k-n-1)f_{jk}+\delta_2\sum_{k=j}^{n}f_{jk}\right)a_{[j]}. \tag{2.37}$$

The objective in eq.(2.37) also has the format of eq.(2.1) with $\xi_j=\delta_1\sum_{k=j}^{n}(2k-n-1)f_{jk}+\delta_2\sum_{k=j}^{n}f_{jk}$ for $j=1,...,n$. Thus we obtain the following result, which both extends and improves the results of Mosheiov [61], who provided an $O(n^3)$ time optimization algorithm to solve the $1\left|p_{[j]}=a_{[j]}j^c\right|\delta_1\sum\sum|C_i-C_j|+\delta_2\sum C_j$ problem.

Theorem 5 *The* $1\left|p_{[j]}=(a_{[j]}+bS_{[j]})j^c\right|\delta_1\sum\sum|C_i-C_j|+\delta_2\sum C_j$ *problem is solvable in* $O(n\log n)$ *time.*

As a corollary, we have the following new result, which establishes the polynomial solvability of the variation of completion time problem with the time-dependent processing time:

Corollary 5 *The* $1\left|p_{[j]}=a_{[j]}+bS_{[j]}\right|\delta_1\sum\sum|C_i-C_j|+\delta_2\sum C_j$ *problem is solvable in* $O(n\log n)$ *time.*

Variation of job waiting times

Consider the $1\left|p_{[j]}=(a_{[j]}+bS_{[j]})j^c\right|\delta_1\sum\sum|W_i-W_j|+\delta_2\sum W_j$ problem, where the *waiting time* of the jth job in the sequence is defined by $W_{[j]}=\sum_{i=1}^{j-1}a_{[j]}$ for $j=1,2,...,n$ [3]. Since $W_{[j]}=C_{[j-1]}$, the argument above for sum and variation of completion times also carries over to this section. The only difference is that $\xi_i=\delta_1\sum_{k=i-1}^{n}(2k-n-1)f_{ik}+\delta_2\sum_{k=j-1}^{n}f_{jk}$ in this case. For the same reason as in the last section, each term is always non-negative. Thus, we obtain the following result:

Theorem 6 *The* $1\left|p_{[j]}=(a_{[j]}+bS_{[j]})j^c\right|\delta_1\sum\sum|W_i-W_j|+\delta_2\sum W_j$ *problem is solvable in* $O(n\log n)$ *time.*

As corollaries, we have the following new results for the corresponding learning effect problem and time-dependent processing time problem.

Corollary 6 *The* $1\left|p_{[j]}=a_{[j]}j^c\right|\delta_1\sum\sum|W_i-W_j|+\delta_2\sum W_j$ *problem is solvable in* $O(n\log n)$ *time.*

Corollary 7 *The* $1\left|p_{[j]}=a_{[j]}+bS_{[j]}\right|\delta_1\sum\sum|W_i-W_j|+\delta_2\sum W_j$ *problem is solvable in* $O(n\log n)$ *time.*

Chapter 3

Scheduling with Learning Effects and/or Time-dependent Processing Times to Minimize the Weighted Number of Tardy Jobs on a Single Machine

Similar to the previous chapter, the general problem we study in this chapter may be stated as follows: n independent, non-preemptive jobs, $J = \{1, 2, ..., n\}$, are available for processing at time zero and are to be processed on a single machine. A schedule is defined by a job sequence $\pi = ([1], [2], ..., [n])$, where $[j]$ represents the job that is in the jth position in π for $j = 1, 2, ..., n$. Our objective is to determine a schedule which minimizes a general unified cost function that is the sum of scheduling costs, expressed by using positional penalties, and tardiness penalties. This unified cost function can be formulated as follows :

$$g(\pi) = \sum_{j=1}^{n} \xi_j \eta_{[j]} + \sum_{j=l+1}^{n} \psi_{[j]}, \tag{3.1}$$

where ξ_j is a *positional, job-independent penalty* for *any* job scheduled in the jth position, and ψ_j is the penalty for job j if it is late, for $j = 1, ..., n$; l is a variable to be decided.

This chapter is organized as follows: In Section 1, we present an $O(n^4)$ optimization algorithm to determine the optimal schedule for the unified problem. In Section 2, we present a group of scheduling problems, where the objective function includes costs for total number of tardy jobs, makespan, and due date assignment. We show that, for all different due date assignment

methods, the objective can be reformulated as a special case of (3.1), which enables us to solve this set of problems in $O(n^4)$ time. Using dynamic programming techniques, we improve this time complexity to $O(n^2)$ for some important special cases in Section 3 and Section 4.

3.1 The Unified Problem

We show in this section that our problem can be solved in $O(n^4)$ time by solving a linear assignment problem. First we consider the case when l is given. For $1 \leq i, j \leq n$, let us define c_{ij} as

$$c_{ij} = \begin{cases} \xi_i \eta_j & \text{for} \quad i \leq l \\ \xi_i \eta_j + \psi_j & \text{for} \quad l+1 \leq i \leq n. \end{cases} \tag{3.2}$$

Our sequencing problem then reduces to the classical linear assignment problem of finding π which minimizes $\sum_{i=1}^{n} c_{i[i]}$. Since the calculation of the assignment costs depends on l, we denote this assignment problem by *P1(l)*. It is well known that a linear assignment problem can be solved in $O(n^3)$ time (see Papadimitriou and Steiglitz [69]).

Therefore, we have the following result:

Lemma 8 *The optimal sequence, denoted by π^*, can be obtained by solving a linear assignment problem requiring $O(n^3)$ time for fixed l.*

The results of our analysis are summarized in the following optimization algorithm:

Algorithm 3 *The optimization algorithm for the solution of the $1 \,||\, \sum \xi_j \eta_{[j]} + \sum \psi_{[j]}$ problem.*

Step 1. Calculate the c_{ij} values by (3.2).

Step 2. Solve the assignment problem *(P1(l))* to determine the optimal job sequence, and denote the resulting optimal sequence by $\pi^* = ([1], [2], ..., [n])$.

If l is not known, then we have to enumerate all l and run Algorithm 3 repeatedly as a subroutine:

Algorithm 4

Initialization: $Z^* = \infty$, $l = 0$.

while $l \leq n$ **do**

Step 1 Apply Algorithm 3. Denote the optimal job sequence by $\pi^*(l) = ([1], [2], ..., [n])$ and the minimum cost by $Z^*(l)$.

Step 2. If $Z^*(l) \leq Z^*$, then set $Z^* = Z^*(l)$, $l^* = l$, $\pi^* = \pi^*(l)$.

Step 3. $l = l + 1$.

end

From Lemma 8, we can get the following result easily:

Theorem 7 *Algorithm 4 solves the* $1 \,||\, \sum \xi_j \eta_{[j]} + \sum \psi_{[j]}$ *problem in* $O(n^4)$ *time.*

In the next sections, we present various applications of our general approach to solve a large set of important scheduling problems with variable processing times.

3.2 Minimizing Total Weighted Number of Tardy Jobs with Due Date Assignment

In this section, we show how our unified method can be used to solve a large set of scheduling problems involving due date assignment decisions.

Similar to the previous chapter, we re-define $g(\pi)$ from (3.1) as $g(\pi, \mathbf{d})$. Cases will be considered by providing special forms of $g(\pi, \mathbf{d})$.

Our objective is to find a job sequence $\pi^* = ([1], [2], ..., [n])$ and a set of due dates $\mathbf{d}^* = (d_1^*, d_2^*, ..., d_n^*)$ to minimize a cost function that includes the costs of tardiness, due date assignment and makespan. For the *CON*, *SLK* and *DIF* due date assignment methods, it is given by

$$g(\pi) = \sum_{j=1}^{n} \psi_j U_j + \gamma \sum_{j=1}^{n} d_j + \delta C_{\max} \tag{3.3}$$

where γ is a nonnegative parameter representing the cost of one unit of due date, U_j is the tardiness indictor (it is equal to 1 if job j is tardy and 0 if otherwise), and δ is a nonnegative parameter representing the cost of one unit of makespan.

It is easy to see that the jobs are processed from time zero and there is no idle time between them, i.e., the condition of Lemma 3 is satisfied for all the due dates assignment methods. Therefore, (2.2) holds in all cases. Next we show that, under an optimal due date assignment strategy, the scheduling cost can be reduced to the format of (3.1) for all of them. For the first time, we present results with different due date assignment methods, including CON, DIF and SLK, to minimize total number of tardy jobs.

In the following subsections, we will reduce the scheduling problem of each due date assignment method to the format of (3.1), therefore they can all be solved in $O(n^4)$ time. For any sequence π, let us define set $E(\pi)$ as the set of on-time jobs under an optimal due date assignment strategy, define the set of tardy jobs by $T(\pi) = J \setminus E(\pi)$. Then the following lemma from [82] is applicable to our problems (we omit the proof since it is rather straightforward):

Lemma 9 *There is an optimal schedule in which the corresponding π sequences $E(\pi)$ before set $T(\pi)$.*

Let $l = |E(\pi)|$ denote the number of early jobs in the optimal schedule, then we have $E(\pi) = \{[1], [2], ..., [l]\}$.

***DIF* due date assignment method**

By Lemma 9, we can schedule all the l early jobs before all the tardy jobs. For job j, if it is early, its due date is at least as large as its completion time. Therefore, the optimal assignment strategy is to let $d^*_{[j]} = C_{[j]}$, for $j = 1, \cdots, l$. On the other hand, if job j is tardy instead, then it is best to set its due date as early as possible, i.e., let $d^*_{[j]} = 0$ for $j = l+1, \cdots, n$. From (2.4) and (3.3), we have

$$\begin{aligned} g(\pi) &= \sum_{j=l+1}^{n} \psi_{[j]} + \gamma \sum_{k=1}^{l} C_{[k]} + \delta C_{[n]} \\ &= \sum_{j=l+1}^{n} \psi_{[j]} + \gamma \sum_{k=1}^{l} \sum_{j=1}^{k} f_{jk} a_{[j]} + \delta \sum_{j=1}^{n} f_{jn} a_{[j]} \\ &= \sum_{j=1}^{l} (\gamma \sum_{k=j}^{l} f_{jk} + \delta f_{jn}) a_{[j]} + \delta \sum_{j=l+1}^{n} f_{jn} a_{[j]} + \sum_{j=l+1}^{n} \psi_{[j]}. \end{aligned}$$

We can see that this is indeed in the format of (3.1), where

$$\xi_j = \begin{cases} \gamma \sum_{k=j}^{l} f_{jk} + \delta f_{jn} & \text{for} \quad j \le l \\ \delta f_{jn} & \text{for} \quad l+1 \le j \le n. \end{cases} \tag{3.4}$$

As in the previous chapter, this equation becomes simpler if there is only learning effect or only time-dependent processing time. In fact, if there is only learning effect, we have $b = 0$, therefore f_{jk} in (2.3) is reduced to j^c for every k. Thus the equation above becomes

$$\xi_j = \begin{cases} \gamma(l-j+1) j^c + \delta j^c & \text{for} \quad j \le l \\ \delta j^c & \text{for} \quad l+1 \le j \le n. \end{cases} \tag{3.5}$$

On the other hand, if there is only time-dependent processing time effect, we have $c = 0$, therefore f_{jk} becomes $(1+b)^{k-j}$. Thus the equation above becomes

$$\xi_j = \begin{cases} \gamma(l-j+1) \sum_{k=j}^{l} (1+b)^{k-j} + \delta(1+b)^{n-j} & \text{for} \quad j \le l \\ \delta(1+b)^{n-j} & \text{for} \quad l+1 \le j \le n. \end{cases} \tag{3.6}$$

***CON* due date assignment method**

It can be easily seen that the optimal strategy is to assign $C_{[l]}$ to all $d^*_{[j]}$. From (2.4) and (3.3), we have

$$\begin{aligned} g(\pi) &= \sum_{j=l+1}^{n} \psi_{[j]} + n\gamma C_{[l]} + \delta C_{[n]} \\ &= \sum_{j=l+1}^{n} \psi_{[j]} + n\gamma \sum_{j=1}^{l} f_{jl} a_{[j]} + \delta \sum_{j=1}^{n} f_{jn} a_{[j]} \\ &= \sum_{j=1}^{l} (n\gamma f_{jl} + \delta f_{jn}) a_{[j]} + \sum_{j=l+1}^{n} \psi_{[j]} + \delta \sum_{j=l+1}^{n} f_{jn} a_{[j]}. \end{aligned}$$

We can see that this is indeed the format of (3.1), where

$$\xi_j = \begin{cases} n\gamma f_{jl} + \delta f_{jn} & \text{for} \quad j \leq l \\ \delta f_{jn} & \text{for} \quad l+1 \leq j \leq n. \end{cases} \tag{3.7}$$

If there is only learning effect, we have $b = 0$, therefore f_{jk} is reduced to j^c for every k. Thus the equation above becomes

$$\xi_j = \begin{cases} n\gamma j^c + \delta j^c & \text{for} \quad j \leq l \\ \delta j^c & \text{for} \quad l+1 \leq j \leq n. \end{cases} \tag{3.8}$$

On the other hand, if there is only time-dependent processing time effect, we have $c = 0$, therefore f_{jk} becomes $(1+b)^{k-j}$. Thus the equation above becomes

$$\xi_j = \begin{cases} n\gamma(1+b)^{l-j} + \delta(1+b)^{n-j} & \text{for} \quad j \leq l \\ \delta(1+b)^{n-j} & \text{for} \quad l+1 \leq j \leq n. \end{cases} \tag{3.9}$$

***SLK* due date assignment method**

Let $h = l-1$. From [83], also as it was shown in Lemma 5, we know that the optimal common slack value is $s^* = C_{[h]}$. From (2.4) and (3.3), we have

$$
\begin{aligned}
g(\pi) &= \sum_{j=l+1}^{n} \psi_{[j]} + \gamma \sum_{j=1}^{n} (s^* + p_{[j]}) + \delta C_{[n]} \\
&= \sum_{j=l+1}^{n} \psi_{[j]} + n\gamma C_{[h]} + \gamma C_{[n]} + \delta C_{[n]} \\
&= \sum_{j=l+1}^{n} \psi_{[j]} + n\gamma \sum_{j=1}^{h} f_{jh} a_{[j]} + (\gamma + \delta) \sum_{j=1}^{n} f_{jn} a_{[j]} \\
&= \sum_{j=1}^{h} (n\gamma f_{jh} + (\gamma + \delta) f_{jn}) a_{[j]} + (\gamma + \delta) \sum_{j=h+1}^{n} f_{jn} a_{[j]} + \sum_{j=l+1}^{n} \psi_{[j]}.
\end{aligned}
$$

We can see that this is indeed of the format of (3.1), where

$$
\xi_j = \begin{cases} n\gamma f_{jh} + (\gamma + \delta) f_{jn} & \text{for} \quad j \le h \\ (\gamma + \delta) f_{jn} & \text{for} \quad h+1 \le j \le n. \end{cases} \tag{3.10}
$$

If there is only learning effect, we have $b = 0$, therefore f_{jk} is reduced to j^c. Thus the equation above becomes

$$
\xi_j = \begin{cases} n\gamma j^c + (\gamma + \delta) j^c & \text{for} \quad j \le h \\ (\gamma + \delta) j^c & \text{for} \quad h+1 \le j \le n. \end{cases} \tag{3.11}
$$

On the other hand, if there is only time-dependent processing time effect, we have $c = 0$, therefore f_{jk} becomes $(1+b)^{k-j}$. Thus the equation above becomes

$$
\xi_j = \begin{cases} n\gamma (1+b)^{h-j} + (\gamma + \delta)(1+b)^{n-j} & \text{for} \quad j \le h \\ (\gamma + \delta)(1+b)^{n-j} & \text{for} \quad h+1 \le j \le n. \end{cases} \tag{3.12}
$$

Application of the Unified Optimization Algorithm According to the analysis above, the objective function (3.3) can be written in the format of (3.1) for all three due date assignment methods, and thus we can present the following optimization algorithm to solve the problem for the earliness-tardiness objective:

Algorithm 5 *A unified optimization algorithm for solving the* $1 \left| p_{[j]} = (a_{[j]} + bS_{[j]}) j^c, X \right| \sum_{j=1}^{n} \psi_j U_j + \gamma \sum_{j=1}^{n} d_j + \delta C_{\max}$ *problems for* $X \in \{CON, SLK, DIF\}$.

Step 1. Apply Algorithm 4 where ξ_j is calculated by eq.(3.7) for the *CON* method, by eq.(3.10) for the *SLK* method, and by eq.(3.4) for the *DIF* method.

Step 2. For the *CON* method, assign $C_{[l^*]}$ to all $d^*_{[j]}$. For the SLK method, assign slack value $s^* = C_{[l^*-1]}$ to all jobs. For the DIF method, let $d^*_{[j]} = C_{[j]}$ for $j = 1, 2, \cdots, l^*$, and $d^*_{[j]} = 0$ for $j = l^* + 1, \cdots, n$.

By theorem 7, we have the following result for this algorithm:

Theorem 8 *Algorithm 5 solves the* $1\left|p_{[j]}=(a_{[j]}+bS_{[j]})j^c,X\right|\sum_{j=1}^{n}\psi_jU_j+\gamma\sum_{j=1}^{n}d_j+\delta C_{\max}$ *problems for* $X\in\{CON,SLK,DIF\}$ *in* $O(n^4)$ *time.*

As special cases, we have the following results, when there is only learning effect, or only time-dependent processing times effect, respectively:

Corollary 8 *Algorithm 5 solves the* $1\left|p_{[j]}=a_{[j]}j^c,X\right|\sum_{j=1}^{n}\psi_jU_j+\gamma\sum_{j=1}^{n}d_j+\delta C_{\max}$ *problems for* $X\in\{CON,SLK,DIF\}$ *in* $O(n^4)$ *time.*

Corollary 9 *Algorithm 5 solves the* $1\left|p_{[j]}=a_{[j]}+bS_{[j]},X\right|\sum_{j=1}^{n}\psi_jU_j+\gamma\sum_{j=1}^{n}d_j+\delta C_{\max}$ *problems for* $X\in\{CON,SLK,DIF\}$ *in* $O(n^4)$ *time.*

It is easy to see that, if instead of makespan, we consider the total completion time in the objective of Theorem 8, i.e., we replace $\delta C_{\max}$ with $\theta\sum C_j$, the same argument holds. The only difference is that in the definition of ξ_j, we use $\theta\sum_{k=j}^{n}f_{jk}$ instead of δf_{jn}. Therefore we also have the following result:

Theorem 9 *The* $1\left|p_{[j]}=(a_{[j]}+bS_{[j]})j^c,X\right|\sum_{j=1}^{n}\psi_jU_j+\gamma\sum_{j=1}^{n}d_j+\theta\sum C_j$ *problems for* $X\in\{CON,SLK,DIF\}$ *are solvable in* $O(n^4)$ *time.*

Again, as special cases, we have the following results, when there is only learning effect, or only time-dependent processing times effect, respectively:

Corollary 10 *The* $1\left|p_{[j]}=a_{[j]}j^c,X\right|\sum_{j=1}^{n}\psi_jU_j+\gamma\sum_{j=1}^{n}d_j+\theta\sum C_j$ *problems for* $X\in\{CON,SLK,DIF\}$ *are solvable in* $O(n^4)$ *time.*

Corollary 11 *The* $1\left|p_{[j]}=a_{[j]}+bS_{[j]},X\right|\sum_{j=1}^{n}\psi_jU_j+\gamma\sum_{j=1}^{n}d_j+\theta\sum C_j$ *problems for* $X\in\{CON,SLK,DIF\}$ *are solvable in* $O(n^4)$ *time.*

In the next two sections, we drop the makespan and the total completion time considerations, i.e., $\delta=0$. The reason to do this is that our recursive relation can only be obtained in this way. The objective function (3.3) now becomes:

$$g(\pi)=\sum_{j=1}^{n}\psi_jU_j+\gamma\sum_{j=1}^{n}d_j \tag{3.13}$$

3.3 Fast Algorithms for Cases with Time-dependent Processing Time Effects Only

In this section, we present quadratic-time algorithms for cases without learning effect ($c = 0$ in (1.3)), using dynamic programming techniques. In this case (2.3) becomes:

$$f_{jk} = (1+b)^{k-j}. \tag{3.14}$$

In this section we assume that $b \geq 0$, i.e., we have deteriorating processing times. Our method is to give the recursive relation between the optimal scheduling cost, which contains a term $a_{[j]}$ times some coefficient. To this end, we need to rewrite $C_{[k]}$ (2.4) and $\sum_{j=1}^{k} C_{[j]}$ in term of $a_{[j]}$ as follows:

$$C_{[k]} = \sum_{j=1}^{k} (1+b)^{k-j} a_{[j]}, \tag{3.15}$$

for the ease of the following summation calculation, we change it to the following form:

$$C_{[j]} = \sum_{i=1}^{j} (1+b)^{j-i} a_{[i]}, \tag{3.16}$$

and

$$\sum_{j=1}^{k} C_{[j]} = \sum_{j=1}^{k} \sum_{i=1}^{j} f_{ij} a_{[i]} = \sum_{j=1}^{k} \sum_{i=j}^{k} f_{ji} a_{[j]} = \sum_{j=1}^{k} \sum_{i=j}^{k} (1+b)^{i-j} a_{[j]} = \sum_{j=1}^{k} \frac{(1+b)^{k-j+1}-1}{b} a_{[j]}, \tag{3.17}$$

where we changed the order of summation for the second equality. Note that each term is always non-negative since we assume that $b \geq 0$.

Since the unit penalty γ is same for each job j from (3.13), and the order of tardy jobs is immaterial, we can assume that the early jobs are always sequenced in SPT order. We sort the jobs in SPT order and decide which job to be early or tardy in a backward fashion. For the sake of simplicity, suppose they are already sorted in this way.

Notice that in (3.15) and(3.17), the coefficient of $a_{[j]}$ is dependent only on the number of early jobs after job j: $k - j$. In fact, a_j has a "weight" of $(1+b)^{k-j}$ (in other words, it contributes $(1+b)^{k-j}$ "times") in the calculation of $C_{[k]}$; it has a "weight" of $\frac{(1+b)^{k-j+1}-1}{b}$ (in other words, it contributes $\frac{(1+b)^{k-j+1}-1}{b}$ "times") in the summation $\sum_{j=1}^{k} C_{[j]}$.

This enables us to design the following dynamic programming. Define $w(j,k)$ to be the minimum cost of scheduling jobs $j, j+1, \cdots, n$ s.t. k of them are early, for $1 \leq j \leq n$ and

$0 \leq k \leq n - j + 1$. We have the following set of boundary conditions for all cases when all jobs are tardy:

$$w(j, 0) = \sum_{i=j}^{n} \psi_{[i]}, \tag{3.18}$$

and another boundary condition when $j = n$:

$$w(n, 1) = \gamma a_n.$$

Now we investigate the recursive functions to calculate $w(j, k)$ for the three due date assignment methods:

DIF If job j is early, then there will be $k - 1$ early jobs among $j + 1, \cdots, n$. The optimal due date for job j is C_j. By (3.13) and (3.17), a_j is counted $\gamma \frac{(1+b)^k - 1}{b}$ "times". Therefore

$$w(j, k) = w(j + 1, k - 1) + \gamma \frac{(1 + b)^k - 1}{b} a_j;$$

otherwise, i.e., when job j is tardy, the optimal due date for job j is 0, thus

$$w(j, k) = w(j + 1, k) + \psi_j.$$

Therefore we have the following recursive equation:

$$w(j, k) = \min\{w(j + 1, k - 1) + \gamma \frac{(1 + b)^k - 1}{b} a_j, w(j + 1, k) + \psi_j\}. \tag{3.19}$$

CON By the due date assignment rule, each job is assigned the same due date, which is the sum of the processing times of all the early jobs. Therefore, if job j is early, then it will be counted n times. Thus

$$w(j, k) = w(j + 1, k - 1) + n\gamma(1 + b)^{k-1} a_j$$

by (3.13) and (3.15), and the fact that there are $k - 1$ early jobs after job j; otherwise, i.e., when job j is tardy,

$$w(j, k) = w(j + 1, k) + \psi_j.$$

Therefore, we have the following recursive equation:

$$w(j, k) = \min\{w(j + 1, k - 1) + n\gamma(1 + b)^{k-1} a_j, w(j + 1, k) + \psi_j\}. \tag{3.20}$$

SLK From [83], we know that the optimal common slack value is $s^* = C_{[l-1]}$, where l is the number of early jobs. In other words, the last early job does not contribute to the due date assignment cost, but the other ones (the first $l-1$) do. If job j is early and $k \geq 2$, then

$$w(j,k) = w(j+1,k-1) + n\gamma(1+b)^{k-1}a_j$$

by (3.13) and (3.15), and the fact that there are $k-1 > 0$ early jobs after job j; if job j is early and $k = 1$, then it is the last early job and thus

$$w(j,k) = w(j+1,k-1);$$

i.e.,

$$w(j,1) = w(j+1,0) = \sum_{i=j+1}^{n} \psi_{[i]};$$

if job j is tardy, we have

$$w(j,k) = w(j+1,k) + \psi_j.$$

Therefore, we have the following recursive equation:

$$w(j,k) = \min\{w(j+1,k-1) + n\gamma(1+b)^{k-1}a_j, w(j+1,k) + \psi_j\}. \tag{3.21}$$

for $k \geq 2$, and (for $k = 1$)

$$w(j,1) = \min\{w(j+1,0), w(j+1,1) + \psi_j\} = \min\{\sum_{i=j+1}^{n} \psi_{[i]}, w(j+1,1) + \psi_j\}. \tag{3.22}$$

From the analysis above, we can present the following algorithms:

Algorithm 6 *The optimization algorithm for the solution of* $1\left|p_{[j]} = a_{[j]} + bS_{[j]}, X\right| \sum_{j=1}^{n} \psi_j U_j + \gamma \sum_{j=1}^{n} d_j$ *problems for* $X \in \{CON, DIF\}$.
Initialization:

$$w(n,1) := \gamma a_n$$

For *j* ***From*** *1* ***To*** *n* ***do***

$$w(j,0) := \sum_{i=j}^{n} \psi_{[i]}$$

Recursion:

For *j* ***From*** *n-1* ***Down To*** *1* ***Do***

For k ***From*** *1* ***To*** *n-j+1* ***Do***
f*or CON:*

$$w(j,k) = \min\{w(j+1,k-1) + n\gamma(1+b)^{k-1}a_j, w(j+1,k) + \psi_j\}$$

f*or DIF:*

$$w(j,k) = \min\{w(j+1,k-1) + \gamma\frac{(1+b)^k - 1}{b}a_j, w(j+1,k) + \psi_j\}$$

Optimal solution value:

$$\min\{w(1,k)|0 \le k \le n\}.$$

end

Algorithm 7 *The optimization algorithm for the solution of* $1\left|p_{[j]} = a_{[j]} + bS_{[j]}, SLK\right|\sum_{j=1}^{n}\psi_j U_j + \gamma\sum_{j=1}^{n} d_j$ *problem.*
Initialization:

$$w(n,1) := \gamma a_n$$

For j ***From*** *1* ***To*** n ***do***

$$w(j,0) := \sum_{i=j}^{n}\psi_{[i]}$$

Recursion:
For j ***From*** *n-1* ***Down To*** *1* ***Do***

$$w(j,1) = \min\{w(j+1,0), w(j+1,1) + \psi_j\} = \min\{\sum_{i=j+1}^{n}\psi_{[i]}, w(j+1,1) + \psi_j\}$$

For j ***From*** *n-1* ***Down To*** *1* ***Do***
For k ***From*** *2* ***To*** *n-j+1* ***Do***

$$w(j,k) = \min\{w(j+1,k-1) + n\gamma(1+b)^{k-1}a_j, w(j+1,k) + \psi_j\}$$

Optimal solution value:

$$\min\{w(1,k)|0 \le k \le n\}.$$

end

In summary, we have the following result:

Theorem 10 *Algorithm 6 solves the* $1\left|p_{[j]}=a_{[j]}+bS_{[j]},X\right|\sum_{j=1}^{n}\psi_jU_j+\gamma\sum_{j=1}^{n}d_j$ *problems for* $X\in\{CON,DIF\}$ *in* $O(n^2)$ *time. Algorithm 7 solves the* $1\left|p_{[j]}=a_{[j]}+bS_{[j]},SLK\right|\sum_{j=1}^{n}\psi_jU_j+\gamma\sum_{j=1}^{n}d_j$ *problem in* $O(n^2)$ *time.*

Proof.

The correctness of the algorithms follows from (3.16)-(3.21). For the time complexity, we note that in both algorithms, the outer recursion j takes $O(n)$ time, and the inner recursion k also takes $O(n)$ time. Therefore, the total time complexity is $O(n^2)$. ■

3.4 Fast Algorithms for Cases with Learning Effect Only

In this section, we present quadratic-time algorithms for cases without time-dependent effect ($b=0$ in (1.3)), or makespan consideration ($\delta=0$), using forward dynamic programming techniques. In this case, we have

$$f_{jk}=j^c. \tag{3.23}$$

from (2.3). As in the previous section, our method is to give the recursive relation for the optimal scheduling costs, which contain a term $a_{[j]}$ times some coefficient. To this end, we need to rewrite $C_{[k]}$ (2.4) in terms of $a_{[j]}$ as follows:

$$C_{[k]}=\sum_{j=1}^{k}j^ca_{[j]}. \tag{3.24}$$

Similar to the discussion in the previous section, we can assume that the early jobs are already sequenced in SPT order. Our goal is to decide which job to be early or tardy in a forward fashion.

Notice that in (3.24), the coefficient of $a_{[j]}$ is dependent only on the number of early jobs before job j. This enables us to design the following forward dynamic programming algorithms.

CON Define $w(j,k)$ to be the minimum cost of scheduling jobs $1,2,\cdots,j$ s.t. k of them are early, for $1\leq k\leq j\leq n$. We have the following set of boundary conditions when jobs $1,2,\cdots,j$ are all tardy:

$$w(j,0)=\sum_{i=1}^{j}\psi_{[i]},$$

for $1\leq j\leq n$; and

$$w(1,1) = a_1.$$

If job j is early, then its processing time gets counted in the (common) due date of every job, i.e., n times, and thus

$$w(j,k) = w(j-1,k-1) + nj^c a_j$$

by (3.13) and (3.24), and the fact that there are $k-1$ early jobs before job j; otherwise, i.e., job j is tardy, we have

$$w(j,k) = w(j-1,k) + \psi_j.$$

Therefore, we have the following recursive equation:

$$w(j,k) = \min\{w(j-1,k-1) + nj^c a_j, w(j-1,k) + \psi_j\}.$$

SLK As in the previous section, from [83], we know that the optimal common slack value is $s^* = C_{[l-1]}$, where l is the number of early jobs. In other words, the last early job does not contribute to the due date assignment cost, but the other ones (the first $l-1$) do. Due to this special property, we do not have the same direct recursive equations as in the CON case. However, we can still utilize those recursive equations to get a set of recursive equations for the SLK case.

Let $W(r)$ be the minimum cost to schedule jobs of $1, 2, \cdots, n$ s.t. job r is the last early one. By the above discussion and definition of $w(j,k)$ in the CON part (in which the due date assignment cost for the k early jobs are calculated recursively), we have

$$W(r) = \min_{0 \le k \le r-1} w(r-1,k) + \sum_{i=r+1}^{n} \psi_i$$

for $1 \le r \le n$ (we define $w(0,0) = 0$, and the minimum cost to schedule jobs $1, 2, \cdots, n$ is given by

$$W = \min_{1 \le r \le n} W(r).$$

From the analysis above, we are ready to present the following algorithm:

Algorithm 8 *The optimization algorithm for the solution of* $1 \left| p_{[j]} = a_{[j]} j^c, X \right| \sum_{j=1}^{n} \psi_j U_j + \gamma \sum_{j=1}^{n} d_j$ *problems for* $X \in \{CON, SLK\}$.
Initialization:

$$w(1,1) := \gamma a_1$$

For j ***From*** *1* ***To*** n ***do***

$$w(j,0) := \sum_{i=i}^{i} \psi_{[i]}$$

Recursion:

For j ***From*** *1* ***To*** n ***Do***

For k ***From*** *0* ***To*** j ***Do***

$$w(j,k) = \min\{w(j-1,k-1) + nj^c a_j, w(j-1,k) + \psi_j\}$$

$$W(j) = \min_{0 \le k \le j-1} w(j-1,k) + \sum_{i=j+1}^{n} \psi_i$$

Optimal solution value:

For CON:

$$\min\{w(1,k) | 0 \le k \le n\}.$$

For SLK:

$$W = \min_{1 \le j \le n} W(j).$$

end

In summary, we have the following result:

Theorem 11 *Algorithm 8 solves the* $1 \left| p_{[j]} = a_{[j]} j^c, X \right| \sum_{j=1}^{n} \psi_j U_j + \gamma \sum_{j=1}^{n} d_j$ *problems for* $X \in \{CON, SLK\}$ *in* $O(n^2)$ *time.*

Proof.

The correctness of the algorithms follows from the analysis before them. For the time complexity, we note that in both algorithms, the outer recursion j takes $O(n)$ time, and the inner recursion k also takes $O(n)$ time. Therefore, the total time complexity is $O(n^2)$. ■

Chapter 4

Bicriteria Scheduling with Batching Deliveries

We consider both the single machine model and the parallel machine model. We are given n jobs $N = \{1, 2, \cdots, n\}$. Our model consists of a production part and a distribution part. In the production part, each job $j \in N$ has a given processing time p_j, sometimes with a due date d_j. The jobs must be executed on a machine without interruption. All jobs and machines are available at time 0. In the distribution part, we assume that there are enough vehicles available.

Given a production schedule, we use the following notations:

C_j: the completion time of job j;

D'_j: the delivery time of job j;

$L_j = D'_j - d_j$: the lateness of job j;

U_j: the indicator of tardiness, i.e., $U_j = 1$ if job j is late, and $U_j = 0$ otherwise;

$P_j = \sum_j p_j$: the total processing time of the first j jobs;

ϕ: the upper bound of batch size, i.e., each vehicle can carry up to ϕ jobs in one shipment, which is assumed to be a given constant;

T: the minimum time between any consecutive deliveries;

f: the transportation cost incurred by each shipment;

z: the number of batches;

Δ: the batch setup time.

We have two objectives to optimize: the scheduling cost and the distribution cost. We use X to denote the scheduling cost, where X may be total delivery time, maximum lateness, or total number of tardy jobs, and $Y = zf$ to measure the distribution cost.

Since our problem has two criteria, we have four different forms of this optimization problem:

(P1) To minimize $\alpha X + (1 - \alpha)Y$, for some $\alpha \in [0, 1]$;

(P2) To minimize X subject to $Y \leq Y'$, where Y' is a given upper bound on the distribution cost;

(P3) To minimize Y subject to $X \leq X'$, where X' is a given upper bound on the delivery cost;

(P4) To identify the set of Pareto-optimal schedules (points) (X, Y), where a schedule S with $X = X(S)$ and $Y = Y(S)$ is called *Pareto-optimal* if there does not exist another schedule S' such that $X(S') \leq X(S)$ and $Y(S') \leq Y(S)$, with at least one of these inequalities being strict. In other words, to construct the *efficiency frontier* or the *trade-off curve*.

Of these four forms, (P1) is the easiest. The optimal solution of any of the other three forms automatically gives the optimal solution of (P1). (P4), on the other hand, is the hardest. Its optimal solution implies the optimal solution of (P1)-(P3).

In many bicriteria scheduling problems, the (P4) form is NP-hard [65]. In this thesis, however, we can solve it in polynomial time, utilizing the special properties of batching.

Chen et al. [15], Hochbaum et al. [42], and Hall et al. [38] [39] all presented dynamic programming methods to solve (P1) in polynomial time, for different objectives X. However, the polynomial-time solvability of (P1) does not imply polynomial-time solvability of (P2), (P3) or (P4).

Notice that the distribution cost Y depends only on z, the number of batches. We will use z to measure the distribution cost instead of Y, for the sake of convenience. In each problem we consider in this paper, we give an optimal solution value for every possible z, the number of batches. The optimal solutions (schedules) can be found with standard backtracking techniques. Some solutions might not be Pareto-optimal. For example, it might happen that both (z, X) and $(z-1, X')$ are solutions given by our algorithms, but $X \geq X'$; therefore, the first one is not Pareto-optimal. In that case, we simply remove it from the solution set.

In the standard three-field notation $A\,|B|\,C$, if ϕ appears in the B section, it means that the upper bound constraint is imposed; if T appears in the B section, it means that the minimum time between consecutive deliveries is imposed. We use (z, X) in the ϕ field to denote the P4-form (Pareto-optimal) objective, where X could be $\sum D'_j$, $\max L_j$, or $\sum U_j$, and the delivery cost is measured by total delivery time, maximum lateness, or total number of tardy jobs, respectively.

We include a straightforward optimality property for all the problems from [15].

Lemma 10 *There exists an optimal schedule where the departure time of each shipment is the completion time of the last job included in the shipment.*

4.1 Minimizing the Total Delivery Time on a Single Machine

We use $X = D'_1 + D'_2 + \cdots + D'_n$ in this section. A set of jobs is in SPT (shortest-processing-time first) order if the jobs are sequenced in a nondecreasing order of their processing times, and jobs with equal processing times are sequenced in the order of their indices. According to this definition, there is exactly one SPT order for a given set of jobs.

We have the following result, which enables us to focus on SPT order in this section:

Lemma 11 *For problems* $1||(z, \sum D'_j)$ *and* $1|\phi, T|(z, \sum D'_j)$*, there exists an optimal schedule in which the jobs are sequenced in SPT order.*

Proof. Consider an optimal schedule σ^*. First we sort the jobs of each batch in SPT order without changing the cost. If the resulting σ^* is not in SPT order, then we must have a job j that is the last job in some batch B, and another job i that is the first job in the next batch B', where $p_j > p_i$. Consider a new schedule σ that is created by exchanging jobs j and i, and forming delivery batches containing jobs $B \cup \{i\} - \{j\}$ and $B' \cup \{j\} - \{i\}$, where both batches are delivered no later than their counterparts in σ^*. All other delivery batches are identical and delivered at the same time in σ as in σ^*. Thus, the cost associated with σ is no more than the cost associated with σ^*. Notice that the number of jobs in each batch remains the same, therefore, the constraint of the batch size is not violated. Also notice that the new delivery time will not violate the T constraint. After a finite number of such exchanges, an optimal schedule is obtained, in which the jobs are sequenced in SPT order. ■

By this lemma, without loss of generality, we can assume in this section that the jobs are indexed in SPT order so that $p_1 \leq \cdots \leq p_n$.

4.1.1 $1|\phi|(z, \sum D'_j)$

We consider the case without transportation timing constraints in this subsection. By Lemma 10 and Lemma 11, the following dynamic programming algorithm finds an optimal schedule for $1|\phi|(z, \sum D'_j)$.

Algorithm 9 *Let* $V(j, z)$ *be the minimum total delivery time of scheduling jobs* $\{1, \cdots, j\}$ *in* z *batches, such that each batch contains at most* ϕ *jobs.*

Initial condition: $V(j, z') = +\infty$ *for* $j > z'c$ *and* $V(j, z') = 0$ *for* $j = 0$*, and* $z' \leq z$*.*

Optimal solution values: $V(n, z)$*;* $z = 1, 2, \cdots, n$*.*

Recursive equation:

$$V(j,z) = \min\{V(j-h, z-1) + hP_j + hz\Delta | h = 1, \cdots, \min(c,j)\}$$

Theorem 12 *Algorithm 9 solves the* $1|\phi|(z, \sum D'_j)$ *problem in* $O(n^2)$ *time.*

Proof. From Lemma 10, we know that the departure time of the last shipment is always $C_j = P_j + z\Delta$, regardless of the size of the last shipment. This implies that the delivery time of all the jobs in the last shipment is $P_j + z\Delta$. In the recursive equation, the value function is computed by trying every possible size of the last shipment. Given that the size of the last shipment is h (i.e., the last shipment delivers jobs $j-h+1, j-h+2, \cdots, j$), the delivery cost contributed by the last shipment is $hC_j = hP_j + hz\Delta$, which is its contribution to the total delivery time. This proves the correctness of the recursive relation and hence the optimality of the algorithm. There are $O(n^2)$ states in this DP, and it takes no more than $O(\phi)$ time to calculate the value for each state. Sorting the jobs in SPT order takes $O(n \log n)$ time. Therefore, the overall time complexity of Algorithm 9 is bounded by $O(n^2)$. ■

4.1.2 $1|\phi, T|(z, \sum D'_j)$

The transportation timing constraint makes the problem more complicated. We need additional variables to keep track of when the next transporter is available. We introduce a state variable u which specifies the last job that is delivered immediately after its processing is completed, and a state variable v that specifies the number of subsequent deliveries. Thus, the transporter performs its last delivery in the (partial) schedule at time $P_u + (z-v)\Delta + vT$, where z is the total number of batches used. We present the following dynamic programming algorithm. Our method is an adaptation from the one in [39].

Algorithm 10 *Let* $V(j,u,v,z)$ *be the minimum total delivery time of scheduling jobs* $\{1, \cdots, j\}$ *in* z *batches, where the last delivery when a job completes processing occurs at time* $P_u + (z-v)\Delta$ *and there are* v *subsequent deliveries, so that the batch containing job* j *is delivered at time* $P_u + (z-v)\Delta + vT$*, where* $v \le j-u, u \le j$*, and* $u = j$ *if* $v = 0$*.*

Initial condition (we require that the first delivery cannot happen before T *here):*

$$V(j,j,0,z') = \begin{cases} 0 & if\ j = 0 \\ jP_j & 0 < j \le c, P_j < T \end{cases}$$

for $z' \le z$*.*

Optimal solution value:

$$\min\{V(n,u,v,z) | 1 \le u \le n, 0 \le v \le z \le n\}$$

.

Recursive equations:

$$V(j,j,0,z) = \min\{V(i,u,v-1,z-1)+(j-i)(P_j+z\Delta) | 0 \le u \le i < j, j-i \le c, 1 \le v \le \min\{i-u+1, j\}\}$$

for $P_j \ge T$, *where* $V = \lfloor (P_j - P_u)/T \rfloor$; *and*

$$V(j,u,v,z) = \min\{V(i,u,v-1,z-1)+(j-i)(P_u+vT+(z-v)\Delta) | 0 \le u \le i < j, j-i \le c, v > V\}.$$

Theorem 13 *Algorithm 10 solves* $1|\phi,T|(z,\sum D'_j)$ *in* $O(n^4)$ *time.*

Proof. The initial condition considers all possible first batches that are delivered before time T. In the first recurrence equation, a delivery occurs at time P_u and then at each T time units until time $P_u + (v-1)T$, and then another delivery for batch $\{i+1, \cdots, j\}$ occurs at time $P_j + z\Delta \ge P_u + (z-v)\Delta) + vT$. The second recurrence equation represents the case where deliveries are previously scheduled at times $P_u, P_u + T, ..., P_u + (v-1)T$, and then batch $\{i+1, \cdots, j\}$ is delivered at time $P_u + (z-v)\Delta) + vT < P_j + z\Delta$. Combined with Lemmas 10 and 11, the correctness of the theorem is proved.

Now let's prove the time complexity. By definition, $j, u, v, z \le n$. Consequently, the first recurrence relation with $u = j$ and $v = 0$ requires $O(\phi n^2)$ time for each j and z. The second recurrence relation requires $O(\phi)$ time for each j, u, z and v. $O(n^2)$ values are calculated in the first recurrence and $O(n^4)$ values are calculated in the second recurrence. Therefore, the overall time complexity of Algorithm 2 is $O(n^4)$. ■

4.2 Minimizing the Maximum Lateness on a Single Machine

We use $X = \max_j L_j$ in this section. A set of jobs are in EDD (earliest-due-date first) order if the jobs are sequenced in a nondecreasing order of their due dates.

When dealing with due-date-related objectives, such as maximum lateness, total number of tardy jobs, or total tardiness, EDD rule is commonly used. However, if the batch size constraint is imposed, EDD is not necessarily optimal. Below is a counter-example for $1||(z, \max_j L_j)$:

There are 4 jobs to be scheduled, where $p_1 = p_2 = p_4 = 1$, $p_3 = 2$, and $d_1 = 2, d_2 = 6, d_3 = d_4 = 5$, and $c = z = 2$. It is easy to verify that the only schedule for each job to be on time is to

schedule job 1 and 2 in the first batch, job 3 and 4 in the second batch. However, the EDD rule is violated in this batch schedule ($d_2 > d_3$, but job 2 is scheduled before job 3.)

Fortunately, we can still apply the EDD rule if there is no batch size constraint. We have the following result, which enables us to focus on EDD order in this section:

Lemma 12 *For problems* $1||(z, \max_j L_j)$ *and* $1|T|(z, \max_j L_j)$*, there exists an optimal schedule in which the jobs are sequenced in EDD order.*

Proof. Consider an optimal schedule σ^*. First, we sort the jobs of each batch in EDD order, without changing the cost. If the resulting σ^* is not in EDD order, then we must have a job j that is the last job in some batch B, and another job i that is the first job in the next batch B', where $d_j > d_i$. Consider a new schedule σ that is created by moving job j from B to B', and forming delivery batches containing jobs $B - \{j\}$ and $B' \cup \{j\}$, where both batches are delivered no later than their counterpart in σ^*. All other delivery batches are identical and delivered at the same time in σ as in σ^*. Only job j is delivered later in σ than in σ^*. However, since jobs j and i are delivered at the same time in σ and $d_j > d_i$, we have $L_j(\sigma) < L_i(\sigma) = L_i(\sigma^*)$, which establishes that σ is another optimal schedule. Thus, the cost associated with σ is no more than the cost associated with σ^*. After a finite number of such exchanges, an optimal schedule is obtained, in which the jobs are sequenced in EDD order. ■

4.2.1 $1||(z, \max_j L_j)$

By Lemma 12, without loss of generality, we assume that the jobs are indexed in EDD order, so that $d_1 \leq \cdots \leq d_n$. The following dynamic programming algorithm finds an optimal schedule for this problem.

Algorithm 11 *Let* $V(j, z)$ *be the minimum delivery cost of scheduling jobs* $\{1, \cdots, j\}$ *in* z *batches.*

Initial condition: $V(j, z') = 0$ *for* $j = 0$*;* $V(j, z') = \infty$ *for* $j = 1, 2, \cdots, n$ *and* $z' \leq z$.

Optimal solution value: $V(n, z)$.

Recursive equation:

$$V(j, z) = \min\{\max(V(j - h, z - 1), P_j + z\Delta - d_{j-h+1}) | h = 1, \cdots, j)\}$$

Theorem 14 *Algorithm 11 finds an optimal schedule for* $1||(z, \max_j L_j)$ *in* $O(n^3)$ *time.*

Proof. From Lemma 10, we know that the departure time of the last shipment is always $C_j = P_j + z\Delta$, regardless of its size. This implies that the delivery time of all the jobs in the

last shipment is $P_j + z\Delta$. In the recursive relation, the value function is computed by trying every possible size of the last shipment. Given that the size of the last shipment is h (i.e., the last shipment delivers jobs $j-h+1, j-h+2, \cdots, j$), the maximum lateness in the last batch is $P_j + z\Delta - d_{j-h+1}$, which is the lateness of job $j-h+1$, by the EDD assumption. This proves the correctness of the recursive relation, and hence, the optimality of the algorithm. There are $O(n^2)$ states in this DP, and it takes no more than $O(n)$ time to calculate the value for each state. Sorting the jobs in EDD order takes $O(n \log n)$ time. Therefore, the overall time complexity of Algorithm 2 is bounded by $O(n^3)$. ■

4.2.2 $1|T|(z, \max_j L_j)$

By Lemma 12, without loss of generality, we also assume that the jobs are indexed in EDD order, so that $d_1 \leq \cdots \leq d_n$. As in subsection 4.1.2, we also introduce additional variables u and v, which have the same meaning. Our method is an adaptation from the one in [39].

Algorithm 12 *Let $V(j,u,v,z)$ be the minimum maximum lateness of scheduling jobs $\{1, \cdots, j\}$ in z batches, where the last delivery when a job completes processing occurs at time $P_u + (z-v)\Delta$ and there are v subsequent deliveries, so that the batch containing job j is delivered at time $P_u + (z-v)\Delta + vT$, where $v \leq j-u, u \leq j$, and $u = j$ if $v = 0$.*

Initial condition:

$$V(j,j,0,1) = P_j - d_1,$$

for $j > 0$ and $P_j < T$.

Optimal solution value:

$$\min\{V(n,u,v,z | 1 \leq u \leq n, 0 \leq v \leq z \leq n\}$$

.

Recursive equations:

$$V(j,j,0,z) = \min\{\max\{V(i,u,v-1,z-1), P_j + z\Delta - d_{i+1}\} | 0 \leq u \leq i < j, 1 \leq v \leq \min\{i-u+1, V\}\}$$

for $P_j \geq T$, where $V = \lfloor (P_j - P_u)/T \rfloor$; and

$$V(j,u,v,z) = \min\{\max\{V(i,u,v-1,z-1), P_u + vT + (z-v)\Delta - d_{i+1}\} | 0 \leq u \leq i < j, v > V\}$$

The interpretation of this algorithm is similar to subsection 4.1.2. In both cases of the recursive relations, the maximum lateness in the last batch $\{i+1, \cdots, j\}$ occurs for the first job $i+1$, which has the earliest due date.

Theorem 15 *Algorithm 12 solves* $1|T|(z, \max_j L_j)$ *in* $O(n^5)$ *time.*

Proof. The initial condition considers all possible first batches that are delivered before time T. In the first recurrence equation, a delivery occurs at time P_u and then at each T time units until time $P_u + (v-1)T$, and then another delivery for batch $\{i+1, \cdots, j\}$ occurs at time $P_j + z\Delta \geq P_u + (z-v)\Delta) + vT$. The second recurrence equation represents the case where deliveries are previously scheduled at times $P_u, P_u + T, ..., P_u + (v-1)T$, and then batch $\{i+1, \cdots, j\}$ is delivered at time $P_j + z\Delta < P_u + (z-v)\Delta) + vT$. Combined with Lemmas 10 and 12, the correctness of the theorem is proved.

By definition, $j, u, v, z \leq n$. Consequently, the first recurrence relation with $u = j$ and $v = 0$ requires $O(n^3)$ time for each j and z . The second recurrence relation requires $O(n)$ time for each j, u, z and v. Therefore, the overall time complexity of Algorithm 12 is $O(n^5)$. ■

4.3 Minimizing the Total Weighted Number of Tardy Jobs on a Single Machine $1||(z, \sum_j w_j U_j)$

In this section, we assume that the late jobs are not delivered. We refer to [85] and [86] for the tardy job delivery case.

We use $X = \sum_j w_j U_j$ in this section. As in the previous section, without loss of generality, we assume that the jobs are indexed in EDD order, so that $d_1 \leq \cdots \leq d_n$.

First we present a pseudo-polynomial time algorithm, then we convert it to a FPTAS.

4.3.1 A Pseudo-polynomial time algorithm for $1||(z, \sum_j w_j U_j)$

Our algorithms in this subsection are adapted from those by Brucker and Kovalyov [9].

Let $W^*(z)$ be the optimal objective value for the problem when the number of early batches is z, and let B be a positive integer, serving as a parameter for our algorithm. In this subsection, we present a dynamic programming algorithm $DP(B)$ which either solves the problem or establishes that $W^*(z) > B$, for $1 \leq z \leq n$.

Our dynamic programming is a forward one. When we work on job j, we have three choices:

- job j is scheduled to be late,

- job j is added as the last one in the last early batch if it can be completed by the earliest due date in that batch,
- job j is assigned to a new batch if it can be completed before its due date d_j.

In our algorithm, the completion time of the last early job is the objective function value, while the weighted number of late jobs $X = \sum_j w_j U_j$ and the earliest due date d in the last early batch are state variables.

Now we state $DP(B)$ formally. Define $C_j(W, d, z)$ as the minimum completion time of the last early job subject to j jobs are scheduled, the weighted number of late jobs is equal to W, there are z number of early batches, and the earliest due date in the last early batch is equal to d. A formal statement of this dynamic programming algorithm is as follows.

Algorithm 13 *DP(B)*

Step 1: Initialization:

Number jobs in EDD order.

Set $C_j(W, d, z) = \infty$ *for all* $j = 0, 1, \cdots, n, z = 1, \cdots, n, W \in \{-w_{max}, -w_{max} + 1, \cdots, B\}$, *where* $w_{max} = \max\{w_j | j = 1, \cdots, n\}$, *and* $d \in \{d_1, \cdots, d_n\}$.

Set $C_0(0, d, z) = 0$ *for all* $d \in \{d_1, \cdots, d_n\}$.

Set $j = 1$.

Step 2:Recursion:

Compute the following for all $W \in \{0, \cdots, B\}$ *and* $d \in \{d_1, \cdots, d_n\}$*:*

$$C_j(W, d, z) = \min \begin{cases} C_{j-1}(W - w_j, d, z) & \\ C_{j-1}(W, d, z) + p_j & \text{if} \quad C_{j-1}(W, d, z) + p_j \le d_j, C_{j-1}(W, d, z) + p_j > 0 \\ K & \text{if} \quad d = d_j, \end{cases} \tag{4.1}$$

where $K = \min\{C_{j-1}(W, i, z-1) + p_j + \Delta | C_{j-1}(W, i, z-1) + p_j + \Delta \le d_j, i \in \{d_1, \cdots, d_{j-1}\}\}$. *If* $\{d_1, \cdots, d_{j-1}\}\}$ *is empty, then we set* $K = \infty$.

If $j = n$, *go to Step 3; otherwise set* $j = j + 1$ *and repeat Step 2.*

Step 3: Optimal solution value:

If $C_n(W, d, z) = \infty$ *for all* $W \in \{0, \cdots, B\}$ *and* $d \in \{d_1, \cdots, d_n\}$, *then* $W^*(z) > B$; *Otherwise, define*

$$W^*(z) = \min\{W|_n(W, d, z) < \infty, W \in \{0, \cdots, B\}, d \in \{d_1, \cdots, d_n\}\},$$

then use standard backtracking techniques to find the corresponding optimal schedule.

The performance of this algorithm is given below:

Theorem 16 *In $O(n^3B)$ time, Algorithm 13 either finds the optimal solution $W^*(z) \leq B$, or finds out that $W^*(z) > B$.*

Proof. Algorithm DP(B) runs in $O(n^3B)$ time because there are n different values of j and z, $B+1$ different values of W and n different values of d. We now prove the correctness of this algorithm.

Note that the three values in the right hand side of the recursion equation correspond to the three possible scheduling choices for each job j. It is easy to see from the general principles of dynamic programming that at Step 2 of the algorithm, all possible objective values $W(z) \leq B$ have been constructed. At Step 3, if $C_n(W, d, z) = \infty$ for all $W \in \{0, \cdots, B\}$ and $d \in \{d_1, \cdots, d_n\}$, then there are no solutions with value $W(z) \leq B$, i.e. $W^*(z) > B$, which proves necessity; On the other hand, if $C_n(W, d, z) < \infty$ for some W and d, then $W^*(z)$ is the optimal objective value among those obtained at Step 2. Thus, the sufficiency is also proved.

■

Apparently, if we choose B to be W_Σ, where $W_\Sigma = \sum w_j$ is the sum of all weights, then $DP(B)$ solves our problem in $O(n^3W_\Sigma)$ time. Therefore, the problem with equal weights is solved by $DP(n)$ in $O(n^4)$ time.

4.3.2 A Fully Polynomial Time Approximation Scheme for $1||(z, \sum_j w_jU_j)$

We now present a fully polynomial approximation scheme for the problem. For our problem in this section, an algorithm A_ϵ is a $(1+\epsilon)$-approximation algorithm iff we have $W(z) < (1+\epsilon)W^*(z)$ for all instances and all z, where $W^*(z)$ represents the optimal solution value when the number of early batches is z and $W(z)$ denotes the value of the solution given by the algorithm. Recall that a family of approximation algorithms $\{A_\epsilon\}$ defines a fully polynomial approximation scheme if, for any $\epsilon > 0$, A_ϵ is a $(1+\epsilon)$-approximation algorithm whose complexity is polynomial in n and $1/\epsilon$.

Consider the problem of minimizing the maximum weight of late jobs $\max\{w_j | j \text{ is late}\}$ to schedule the jobs in z batches. Let $W^0(z)$ (we will show how to compute it later) be the minimal solution value of this problem and let I_z^0 be the set of late jobs in the corresponding schedule. If I_z^0 is empty, then this schedule is optimal for the original problem and we have $W^*(z) = 0$. Suppose that I_z^0 is not empty. Then consequently we have $W^0(z) > 0$ and $W^*(z) > 0$. Let I_z^* be the set of late jobs in an optimal schedule for the original problem with z batches. Then the following chain of relations holds:

$$W^0(z) \leq \max\{w_j | j \in I_z^*\} \leq \sum_{j \in I_z^*} w_j = W^*(z) \leq \sum_{j \in I_z^0} w_j \leq nW^0(z).$$

Thus we have $W^0(z) \leq W^*(z) \leq nW^0(z)$. We set $\delta' = \epsilon W^0(z)/n$ and define scaled weights $w'_j = \lfloor w_j/\delta' \rfloor$ for $j = 1, 2, \cdots, n$. We now define a procedure A_ϵ as follows. We apply $DP(n^2/\epsilon)$ to our problem with scaled weights, which provides an optimal solution value $V'(z) = \sum_{j \in I_z} w'_j$ where I_z is the set of late jobs. A_ϵ chooses I_z as a solution for the original problem, i.e. the solution value of A_ϵ is $W'(z) = \sum_{j \in I_z} w_j$. Using the inequalities $\lfloor x \rfloor \leq x < \lfloor x \rfloor + 1$, we have

$$V'(z) \leq \frac{W^*(z)}{\delta'} \leq \frac{nW^0(z)}{\delta'},$$

and

$$W'(z) \leq \delta' V'(z) + n\delta' \leq W^*(z) + n\delta' \leq (1+\epsilon)W^*(z).$$

The inequalities above show that A_ϵ is a $(1+\epsilon)$-approximation algorithm for the original problem and the former inequalities show that $nW^0(z)/\delta' = n^2/\epsilon$ is a valid upper bound for the optimal value $V'(z)$ of the scaled problem. This implies that the running time of A_ϵ is $O(n^5/\epsilon)$. Therefore, the family of algorithms $\{A_\epsilon\}$ forms a fully polynomial approximation scheme for all z.

What remains to do is to show how to calculate $W^0(z)$. Let $(i_l, \cdots, i_n)$ be a sequence of jobs in nonincreasing order of their weights: $w_{i_l} \geq \cdots \geq w_{i_n}$. We can see that $W^0(z) = w_{i_{m+1}}$ where m is the maximum index k for which all jobs $i_l, \cdots, i_k$ can be scheduled early in z batches. If $m = n$ then $W^0(z) = 0$.

To find the index m, we can perform a binary search in the range $1, \cdots, n$, where for each trial value k we ask the question "can all jobs $i_l, \cdots, i_k$ be scheduled early in z batches?". It is shown by Hochbaum and Landy [42] that this question can be answered in $O(n)$ time as follows.

Renumber the jobs so that $d_{i_l} \leq \cdots \leq d_{i_k}$. If we sort the jobs according to nondecreasing d_j-values in a preprocessing step then this renumbering can be done in $O(n)$ time. Schedule jobs in order $1, \cdots, k$ to the end of the current schedule. Assume that $j-1$ jobs have already been assigned. If jobs $1, \cdots, j$ can be scheduled early by adding j to the last batch, then do so. Otherwise, if jobs $1, \cdots, j$ can be early by starting a new batch containing only job j, and the total number of batches does not exceed z, then do so. If neither, then we can conclude that there is no schedule in which jobs $1, \cdots, k$ can be early in z batches. By maintaining the value of the completion time of the last job and the value of the earliest due date in the last batch, each iteration of this recursive subroutine can be done in constant time. Since the number

of iterations of this procedure is at most k, the question can be answered in linear time in n. Therefore, we establish the following result:

Proposition 1 *The problem of minimizing the maximum weight of late jobs such that there are z early batches can be solved in $O(n \log n)$ time for any fixed z.*

Proof. Sorting n jobs by their weights can be done in $O(n \log n)$ time. The binary search for the value of m requires $O(\log n)$ steps, where each step can be done in $O(n)$ time by the analysis above. Thus, the overall time complexity is $O(n \log n)$. ■

We can summarize the above results in the following theorem:

Theorem 17 *A_ϵ solves the problem $1||(z, \sum_j w_j U_j)$ in $O(n^5/\epsilon)$ time. Therefore, $\{A_\epsilon\}$ forms a fully polynomial approximation scheme for all z for the problem $1||(z, \sum_j w_j U_j)$.*

4.4 Minimizing the Total Delivery Time on Two Parallel Machines $P2||\sum D'_j$

We change our subject from single machine to uniform parallel machines in this section. We consider two machines only, but our algorithms extend to any fixed number of machines. Similar to the argument in the case of a single machine, we may assume that the jobs are indexed in SPT order.

Let q_1 and q_2 be the processing speed of machine 1 and 2, respectively. Then the processing time of job j on machine 1 is p_j/q_1, and p_j/q_2 on machine 2. We can rewrite the total delivery time $\sum D'_j$ on machine 1 in terms of p_j and q_1 as follows: given a batching schedule, for job j, let w_j be the number of jobs within or after the batch which j belongs to, then the processing time of job j is counted w_j times in $\sum D'_j$. Therefore the total delivery time on machine 1 is given by

$$\sum_j D'_j = \sum_j \frac{w_j p_j}{q_1}.$$

The same equality holds for machine 2, except that q_1 is replaced with q_2. Define $V(j, k, z)$ to be the minimum total delivery time of scheduling jobs $n-j-k+1, n-j-k+2, \cdots, n$, which are the $j+k$ longest ones of jobs $1, 2, \cdots, n$, in z batches where j jobs of them are on machine 1, and k jobs are on machine 2.

Our dynamic programming algorithm works backwards. From the argument above, we know that the delivery time contributed by each job in the last batch on machine 1 is j times its processing time, and k times on machine 2. In other words, each job in the last batch on machine

1 is counted j/q_1 times, and k/q_2 times on machine 2. Jobs in other batches are counted fewer times than those in the first batch on the same machine. To minimize the total delivery time, by the Hardy-Littlewood-Polya inequality, we need to put the shortest jobs in the first batch on machine 1, if $j/q_1 > k/q_2$; and in the first batch on machine 2, if otherwise. Longer jobs will be put in later batches. Therefore, we have the following algorithm:

Algorithm 14 *Initial condition:*

$$V(j, 0, z') = V(0, k, z') = +\infty$$

for $j > z'c$ and $k > z'c$ (recall that ϕ is the upper bound of the batch size) for $z' \leq z$;

$$V(j, 0, 1) = \Delta + \sum_{l=n-j+1}^{n} p_l/q_1$$

for $j \leq c$; and

$$V(0, k, 1) = \Delta + \sum_{l=n-k+1}^{n} p_l/q_2$$

for $k \leq c$.

Optimal solution value:

$$\min\{V(j, k, z) | j + k = n\}.$$

Recursive equation:

$$V(j,k,z) = \begin{cases} \min_{1\leq i\leq \min\{c,j\}}\{V(j-i,k,z-1) + \Delta + j\sum_{l=n-j-k+1}^{n-j-k+i} p_l/q_1\} & if\ j/q_1 > k/q_2 \\ \min_{1\leq i\leq \min\{c,k\}}\{V(j,k-i,z-1) + \Delta + k\sum_{l=n-j-k+1}^{n-j-k+i} p_l/q_2\} & otherwise \end{cases}$$

Here we try every possible size i of the last batch on both machines, and add its total processing time of the jobs, which are the i shortest among jobs $n - i - j + 1, \cdots, n$, in this batch.

Theorem 18 *Algorithm 14 finds an optimal schedule for $P2||\sum D'_j$ in $O(n^3)$ time.*

Proof. The proof of optimality follows from Lemmas 10 and 11, and the explanation of the recursive equation above Algorithm 14. By definition, $j, k, z \leq n$. Consequently, there are $O(n^3)$ states. Each recursion takes on $O(\phi)$. Therefore, the overall time complexity of Algorithm 2 is $O(n^3)$. ■

Note: There is another interpretation of our model. Instead of parallel machines, we can consider the machines as different suppliers to the same client (such as a manufacturer).

Chapter 5

Bicriteria Scheduling with Batch Deliveries and Due Date Assignments on a Single Machine

In this chapter, we study supply chain scheduling problems with the DIF assumption where arbitrary due dates are allowed to be assigned to jobs individually.

Shabtay and Steiner [82] studied a single-machine scheduling problem, in which each job has a contracted due date and can be given an arbitrary assigned due date. Their goal is to find a schedule which minimizes the sum of due-date-assignment costs and the weighted number of tardy jobs with respect to the assigned due dates, but their model does not include batching or delivery costs. They provide a strong $\mathcal{NP}$-hardness proof for the general case and present polynomial algorithms for two special cases: one with zero contracted due date and a uniform due-date-assignment cost for all jobs and one with uniform contracted due date, equal due-date-assignment costs and equal tardiness penalties (weights) for all jobs. Our problem is essentially the bicriteria version of their problem.

We use A_j for the *contracted due date* for job j, which is the original due date required by the customer. Let D_j denote the assigned due date of job j. Let $R_j = \max\{D_j - A_j, 0\}$ be the extended time units on A_j, and $\lambda_j R_j$ be the due-date-assignment cost, where λ_j is the due-date-assignment cost per extended time unit. If all A_j are equal, then we use notation A to represent the common contracted due date, i.e., $A_j = A$. Let w_j be the tardiness penalty and C_j be the batch completion time of job j.

In the standard three-field notation $A\,|B|\,C$, we use $(z, \sum \lambda_j R_j + \sum w_j U_j)$ in the C field to denote the P4-form (Pareto-optimal) objective, and we add A_j and DIF to the B field to show the contracted due date and the distinct due date assignment method, respectively. We assume

that the batch setup time is Δ.

Our goal is to find a schedule which is Pareto-optimal in the sense that one criterion is to minimize the sum of the due-date-assignment costs and the weighted number of tardy jobs, and the other criterion is to minimize the batch-delivery costs, denoted by $1|A,\text{DIF}|(z, \sum \lambda_j R_j + \sum w_j U_j)$. There are three levels of decisions to be made in this problem:

(1) determining a job sequence;

(2) grouping the sequence into batches;

(3) assigning due dates to each job individually.

Suppose that we are given a schedule σ, where the first two scheduling decisions have been made but no due date has been assigned to any job yet. Since we know the sequence and the batches, we know the number of batches $z(\sigma)$ and the batch-delivery costs $z(\sigma)q$. We need to determine $D_j(\sigma)$ for each j, which minimizes the cost $\sum \lambda_j R_j(\sigma) + \sum w_j U_j(\sigma)$.

The rest of this chapter is organized as follows. In the first section, we study the case with a uniform due-date-assignment cost. We show that it is NP-hard only in the ordinary sense by presenting a pseudo-polynomial algorithm, which requires only polynomial time when all processing times are equal. In the next section, we present a polynomial algorithm for the case with equal due-date-assignment costs and equal tardiness penalties.

5.1 The Problem with Uniform Due Date Assignment Costs

When all due date assignment costs are equal, i.e., $\lambda_j = \lambda$ for all j, we have the following proposition first developed by Shabtay and Steiner [82].

Proposition 2 *For any given schedule σ, the optimal due date assignment is*

$$D_j(\sigma) = \begin{cases} A, & \text{if } C_j(\sigma) \le A \text{ or } C_j(\sigma) > A + \frac{w_j}{\lambda} \\ C_j(\sigma), & \text{if } A < C_j(\sigma) \le A + \frac{w_j}{\lambda} \end{cases} . \tag{5.1}$$

Proposition 2 also implies the following useful observation.

Proposition 3 *For the $1|A,DIF|\sum \lambda R_j + \sum w_j U_j + zq$ problem, there is an optimal schedule in which $A + \frac{w_j}{\lambda}$ is an upper bound for the assigned due date of each job $j \in J$. Furthermore, job j is tardy, with an assigned due date $D_j(\sigma) = A$, in an optimal schedule σ if and only if $C_j(\sigma) > A + \frac{w_j}{\lambda}$.*

We consider $1|A, DIF|(z, \sum \lambda R_j + \sum w_j U_j)$ in this section. It is known that the problem $1|A, DIF|\sum \lambda R_j + \sum w_j U_j + zq$, where q is the cost for each batch, is NP-hard in the ordinary

sense. Therefore, the best we can hope for its bicriteria version is to find a pseudo-polynomial algorithm.

Before presenting such an algorithm, we have the following properties for optimal schedules:

Lemma 13 *There is an optimal schedule for the* $1|A, DIF|(z, \sum \lambda R_j + \sum w_j U_j)$ *problem where all tardy jobs are delivered at the end in a single batch, either by themselves or together with some early jobs.*

Proof. Moving tardy jobs from earlier batches to the last batch never increases the cost. ■

Lemma 14 *There is an optimal schedule for the* $1|A, DIF|(z, \sum \lambda R_j + \sum w_j U_j)$ *problem where all early jobs are sequenced in SPT order.*

Proof.

Suppose otherwise, i.e., there exist two jobs i and k with $p_i > p_k$ that are early in consecutive batches in schedule σ with batch-completion times $C_i(\sigma) < C_k(\sigma)$. By Propositions 2 and 3, we can assume that $D_i(\sigma) \in \{A, C_i(\sigma)\}$, $D_k(\sigma) \in \{A, C_k(\sigma)\}$, $C_i(\sigma) \leq \frac{w_i}{\lambda} + A$ and $C_k(\sigma) \leq \frac{w_k}{\lambda} + A$. Let σ' be a new schedule in which we exchange jobs i and k. Then we have $C_i(\sigma') = C_k(\sigma)$ and $C_k(\sigma') = C_i(\sigma) - p_i + p_k < C_i(\sigma)$. It is apparent to see that all early jobs in σ (excluding job i) remain early after such an exchange. If $C_i(\sigma') \leq \frac{w_i}{\lambda} + A$, then we can let job i be early. In this case, the due-date-assignment cost will not increase because job i and job k have the same λ. If $C_i(\sigma') > \frac{w_i}{\lambda} + A$, then it is better to set $D_i(\sigma') = A$ and let job i be tardy. In this situation, the cost will be reduced by at least

$$\begin{aligned}
&\lambda[D_i(\sigma) + D_k(\sigma)] - \lambda[D_i(\sigma') + D_k(\sigma')] - w_i \\
&= \lambda[\max\{A, C_i(\sigma)\} - \max\{A, C_k(\sigma')\}] + [\lambda \max\{A, C_k(\sigma)\} - \lambda A - w_i] \\
&\geq \lambda[\max\{A, C_i(\sigma)\} - \max\{A, C_k(\sigma')\}] + [\lambda \max\{A, C_k(\sigma)\} - \lambda C_i(\sigma')] \\
&\geq 0.
\end{aligned}$$

Note that the number of jobs in every batch stays the same after we make the exchange mentioned above. After we complete such exchanges for all such pairs in consecutive batches, we can obtain the desired SPT sequence for all early jobs. ■

Algorithm 15 *Let* $V(j, i, z, t)$ *be the minimum cost for processing and delivering jobs* $1, 2, \cdots, j$ *using* z *deliveries, where the completion time of the last early batch is* t *and there are* i *jobs in that batch.*

Initial condition:

$$V(0, 0, z', t) = 0$$

for $z' \leq z$.

Optimal solution value:

$$\min\{V(n,i,z,t)|0 \leq t \leq P, 0 \leq i \leq n\}$$

.

Recursive equations:

$$V(j,i,z,t) = \min \begin{cases} V(j-1,i,z,t) + w_j \\ V(j-1,i,z-1,t-p_j) + \Delta + \lambda \max\{0, t-A\} \ if \ t \leq A + \frac{w_j}{\lambda} \\ V(j-1,i-1,z,t-p_j) + \lambda i \max\{0,t-A\} - \lambda(i-1)\max\{0,t-p_j-A\} \\ \quad if \ t \leq A + \frac{w_j}{\lambda} \end{cases}$$

In the recurrence equation, the first row corresponds to the case when job j is tardy and delivered in the last batch; the second row holds when job j starts a new (early) batch; while the last row is for the case when job j is appended to the last early batch.

Theorem 19 *Algorithm 15 solves the problem $1|A, DIF|(z, \sum \lambda R_j + \sum w_j U_j)$ in $O(n^3 P)$ time.*

Proof. The correctness follows from the explanations of the recurrence. There are $n^3 P$ number of states, and each recurrence takes constant time. Therefore, the total time complexity is $O(n^3 P)$. ■

5.2 The Problem with Uniform Due Date Assignment and Tardy Costs

We consider $1|A, DIF|(z, \sum \lambda R_j + \sum wU_j)$ in this section.

Lemma 13 and Lemma 14 still hold in this section, besides, we have the following properties for optimal schedules:

Lemma 15 *There is an optimal schedule for the $1|A, DIF|(z, \sum \lambda R_j + \sum wU_j)$ problem where all early jobs are scheduled to finish before or at $w/\lambda + A$.*

Proof. $w/\lambda + A$ is an upper bound for any assigned due date. ■

Lemma 16 *There is an optimal schedule for the $1|A, DIF|(z, \sum \lambda R_j + \sum wU_j)$ problem where the early jobs are the shortest jobs.*

Proof.

It follows from Proposition 3 and Lemma 13 that there exists an optimal schedule where the early jobs are delivered in batches before or at $\frac{w}{\lambda} + A$ and the tardy jobs (if any) are delivered after $\frac{w}{\lambda} + A$ in a single batch. Suppose that there are an early job i and a tardy job k with $p_i > p_k$ in a schedule. Then exchanging the position of job i and k may reduce the schedule cost by $\lambda(p_i - p_k)$ and can not lead to an increased cost. Exchanging all such pairs will generate an early job set as claimed. ■

By Lemmas 13-16, we can design the following dynamic programming algorithm for the problem $1|A, DIF|(z, \sum \lambda R_j + \sum wU_j)$:

Algorithm 16 *Let $V(j, z)$ be the minimum cost for processing and delivering early jobs $1, 2, \cdots, j$ using z deliveries, for $P_j + z\Delta = p_1 + p_2 + \cdots + p_j + z\Delta \leq A + \frac{w}{\lambda}$. Let $V'(j, z)$ be the minimum cost for processing and delivering jobs $1, 2, \cdots, n$ using z early deliveries, where the last early job is j.*

Initial condition:

$$V(0, z') = 0$$

for all $z' \leq z$.

Optimal solution value:

$$\min\{V'(j, z) | 0 \leq j \leq n\}$$

.

Recursive equations:

$$V(j, z) = \min_k V(j - k, z - 1) + \lambda k \max\{P_j + z\Delta - A, 0\},$$

for $P_j + z\Delta \leq A + \frac{w}{\lambda}$, and

$$V'(j, z) = V(j, z) + w(n - j).$$

In the recurrence equations, for $V(j, z)$, we try every possible size of the last batch; For $V'(j, z)$, since job j is the last early job, all jobs $j + 1, \cdots, n$ are tardy. Therefore, their tardy costs are added in the equation.

Theorem 20 *Algorithm 16 solves the problem $1|A, DIF|(z, \sum \lambda R_j + \sum wU_j)$ in $O(n^3)$ time.*

Proof. The correctness follows from the explanations of the recurrence. There are n^2 number of states, and each recurrence takes $O(n)$ time . Therefore, the total time complexity is $O(n^3)$. ■

Chapter 6

Summary and Future Research

Our main contributions include using a unified framework to tackle an extensive range of scheduling problems with learning/deteriorating effect Generalize many previous results; improving the time complexity of some previous algorithms; and investigating scheduling problems with batching from a multi-criteria perspective.

In Chapter 2, we presented a very simple and fast solution algorithm for a large number of unrelated scheduling problems using a common product matrix representation of their cost functions. Table 6.1 contains a summary of these results. Similar solutions, which exploit the special structure of the underlying linear assignment problems, were presented in [55] and [48] for certain related scheduling problems with some convex controllable processing times. Somewhat surprisingly, the fastest known solutions for these problems with linear control functions still require $O(n^3)$ time, since they lead to general linear assignment models. It remains a topic for future research whether these latter problems can also be solved by faster assignment algorithms.

Our future research for Chapter 2 is to solve $1 \left| p_{[j]} = (a_{[j]} + bS_{[j]})j^c \right| \sum w_j C_j$ in polynomial time, or prove that it is NP-hard.

Table 6.2 contains a summary of our results on total number of tardy jobs objective from Chapter 3.

It remains an open problem whether there exists a $O(n^2)$ time algorithm for the DIF due date assignment method with learning effect only. Another challenge is to design a $O(n^2)$ time algorithm for the case of $\delta > 0$, i.e., to consider the makespan.

It is interesting to note that by considering the learning/deteriorating effect, we do not have to sacrifice the speed of our algorithms. In fact, our time complexity is the same as the corresponding problems without learning/deteriorating effect. We consider only the simplest form of learning/deteriorating effect. It is another interesting topic for Chapter 2 & 3 as a future research.

Problem	Complexity	Ref.
$1 \left\vert p_{[j]} = (a_{[j]} + bS_{[j]})j^c, CON \right\vert \alpha \sum E_j + \beta \sum T_j + \gamma \sum d_j + \delta C_{\max}$	$O(n \log n)$	Theorem 1
$1 \left\vert p_{[j]} = (a_{[j]} + bS_{[j]})j^c, SLK \right\vert \alpha \sum E_j + \beta \sum T_j + \gamma \sum d_j + \delta C_{\max}$	$O(n \log n)$	Theorem 1
$1 \left\vert p_{[j]} = (a_{[j]} + bS_{[j]})j^c, DIF \right\vert \alpha \sum E_j + \beta \sum T_j + \gamma \sum d_j + \delta C_{\max}$	$O(n \log n)$	Theorem 1
$1 \left\vert p_{[j]} = (a_{[j]} + bS_{[j]})j^c, CONW \right\vert \alpha \sum E_j + \beta \sum T_j + n(\gamma_1 d + \gamma_2 D) + \delta C_{\max}$	$O(n \log n)$	Theorem 1
$1 \left\vert p_{[j]} = (a_{[j]} + bS_{[j]})j^c, CON \right\vert \alpha \sum E_j + \beta \sum T_j + \gamma \sum d_j + \theta \sum C_j$	$O(n \log n)$	Theorem 2
$1 \left\vert p_{[j]} = (a_{[j]} + bS_{[j]})j^c, SLK \right\vert \alpha \sum E_j + \beta \sum T_j + \gamma \sum d_j + \theta \sum C_j$	$O(n \log n)$	Theorem 2
$1 \left\vert p_{[j]} = (a_{[j]} + bS_{[j]})j^c, DIF \right\vert \alpha \sum E_j + \beta \sum T_j + \gamma \sum d_j + \theta \sum C_j$	$O(n \log n)$	Theorem 2
$1 \left\vert p_{[j]} = (a_{[j]} + bS_{[j]})j^c, CONW \right\vert \alpha \sum E_j + \beta \sum T_j + n(\gamma_1 d + \gamma_2 D) + \theta \sum C_j$	$O(n \log n)$	Theorem 2
$1 \left\vert p_{[j]} = (a_{[j]} + bS_{[j]})j^c \right\vert C_{\max}$	$O(n \log n)$	Theorem 3
$1 \left\vert p_{[j]} = (a_{[j]} + bS_{[j]})j^c \right\vert \sum C_j$	$O(n \log n)$	Theorem 4
$1 \left\vert p_{[j]} = (a_{[j]} + bS_{[j]})j^c \right\vert \delta_1 \sum \sum \lvert C_i - C_j \rvert + \delta_2 \sum C_j$	$O(n \log n)$	Theorem 5
$1 \left\vert p_{[j]} = (a_{[j]} + bS_{[j]})j^c \right\vert \delta_1 \sum \sum \lvert W_i - W_j \rvert + \delta_2 \sum W_j$	$O(n \log n)$	Theorem 6

Table 6.1: Summary of results in Chapter 2

We summarize our results on bicriteria scheduling problems with batching from Chapter 4 & 5 in Table 6.3

Our future research for Chapter 4 &5 is to investigate the total tardiness objective.

Problem	Complexity	Ref.
$1 \left\vert p_{[j]} = (a_{[j]} + bS_{[j]})j^c, CON \right\vert \sum \psi_j U_j + \gamma \sum d_j + \delta C_{\max}$	$O(n^4)$	Theorem 8
$1 \left\vert p_{[j]} = (a_{[j]} + bS_{[j]})j^c, SLK \right\vert \sum \psi_j U_j + \gamma \sum d_j + \delta C_{\max}$	$O(n^4)$	Theorem 8
$1 \left\vert p_{[j]} = (a_{[j]} + bS_{[j]})j^c, DIF \right\vert \sum \psi_j U_j + \gamma \sum d_j + \delta C_{\max}$	$O(n^4)$	Theorem 8
$1 \left\vert p_{[j]} = (a_{[j]} + bS_{[j]})j^c, CON \right\vert \sum \psi_j U_j + \gamma \sum d_j + \theta \sum C_j$	$O(n^4)$	Theorem 9
$1 \left\vert p_{[j]} = (a_{[j]} + bS_{[j]})j^c, SLK \right\vert \sum \psi_j U_j + \gamma \sum d_j + \theta \sum C_j$	$O(n^4)$	Theorem 9
$1 \left\vert p_{[j]} = (a_{[j]} + bS_{[j]})j^c, DIF \right\vert \sum \psi_j U_j + \gamma \sum d_j + \theta \sum C_j$	$O(n^4)$	Theorem 9
$1 \left\vert p_{[j]} = a_{[j]} + bS_{[j]}, CON \right\vert \sum \psi_j U_j + \gamma \sum d_j$	$O(n^2)$	Theorem 10
$1 \left\vert p_{[j]} = p_{[j]} = a_{[j]} + bS_{[j]}, DIF \right\vert \sum \psi_j U_j + \gamma \sum d_j$	$O(n^2)$	Theorem 10
$1 \left\vert p_{[j]} = p_{[j]} = a_{[j]} + bS_{[j]}, SLK \right\vert \sum \psi_j U_j + \gamma \sum d_j$	$O(n^2)$	Theorem 10
$1 \left\vert p_{[j]} = a_{[j]}j^c, CON \right\vert \sum \psi_j U_j + \gamma \sum d_j$	$O(n^2)$	Theorem 11
$1 \left\vert p_{[j]} = a_{[j]}j^c, SLK \right\vert \sum \psi_j U_j + \gamma \sum d_j$	$O(n^2)$	Theorem 11

Table 6.2: Summary of results in Chapter 3

Problem	Complexity	Ref.
$1 \left\vert c \right\vert (z, \sum D'_j)$	$O(n^2)$	Theorem 12
$1 \left\vert c, T \right\vert (z, \sum D'_j)$	$O(n^4)$	Theorem 13
$1 \vert\vert (z, \max_j L_j)$	$O(n^3)$	Theorem 14
$1 \vert T \vert (z, \max_j L_j)$	$O(n^5)$	Theorem 15
$1 \vert\vert (z, \sum_j U_j)$	$O(n^3 B)$	Theorem 16
$1 \vert\vert (z, \sum_j w_j U_j)$	$O(n^5/\epsilon)$	Theorem 17
$P2 \left\vert c \right\vert (z, \sum D'_j)$	$O(n^3)$	Theorem 18
$1 \vert A, DIF \vert (z, \sum \lambda R_j + \sum w_j U_j)$	$O(n^3 P)$	Theorem 19
$1 \vert A, DIF \vert (z, \sum \lambda R_j + \sum w U_j)$	$O(n^3)$	Theorem 20

Table 6.3: Summary of results in Chapter 4 and 5

Bibliography

[1] Adamopoulos, G.I. and Pappis, C.P., (1996), Single Machine Scheduling with Flow Allowances, *J. of the Oper. Research Society*, **47**, 1280-1285.

[2] Agnetis, A., Hall, N.G. and Pacciarelli, D., (2006), Supply chain scheduling: Sequence coordination. *Discrete Applied Mathematics*, 154(15):2044–2063.

[3] Bagchi, U.B., (1989), Simultaneous Minimization of Mean and Variation of Flow-Time and Waiting Time in Single Machine Systems, *Oper. Research*, **37**, 118-125.

[4] Baker, K.R. and Scudder, G.D., (1990), Sequencing With Earliness and Tardiness Penalties: A Review, *Oper. Research,* **38**, 22-36.

[5] Biskup D., (1999), Single-machine scheduling with learning considerations, *European Journal of Operational Research*, **115**: 173-178.

[6] Biskup D., (2008), A state-of-the-art review on scheduling with learning effects, *European Journal of Operational Research*, **188**, 315-329.

[7] Browne S. and Yechiali U., (1990), Scheduling Deteriorating Jobs on a Single Processor, *Operations Research*, **38**, 495-498.

[8] Burkard, R.E., Klinz, B., Rudolf, R., (1996), Perspectives of Monge properties in optimization, *Discrete Applied Mathematics*, **70**, 95-161.

[9] Brucker, P. and M.Y. Kovalyov, (1996), Single machine batch scheduling to minimize the weighted number of late jobs. *Mathematical Methods of Operations Research*, 43:1-8.

[10] Brucker, P., (2001), *Scheduling Algorithms*. Springer-Verlag New York, Inc., 3rd edition.

[11] Cheng, T.C.E. and Kahlbacher, H.G., (1993), Scheduling with delivery and earliness penalties. *Asia-Pacific Journal of Operational Research*, 10:145-152.

[12] Cheng, T.C.E. and Kovalyov, M.Y., (1996), Batch scheduling and common due-date assignment on a single machine. *Discrete Applied Mathematics*, 70(3):231-245.

[13] Chen, Z.-L., (1996), Scheduling and common due date assignment with earliness-tardiness penalties and batch delivery costs. *European Journal of Operational Research*, 93(1):49-60.

[14] Chen, B., Potts, C.N. and Woeginger, G.J., (1998), A review of machine scheduling: Complexity, algorithms and approximability. *In Handbook of Combinatorial Optimization, Edited by D.-Z. Du and P.M. Pardalos*, 3:21-169.

[15] Chen, Z.-L. and Vairaktarakis, G.L., (2005), Integrated scheduling of production and distribution operations. *Management Science*, 51(4):614-628.

[16] Chen, Z.-L. and Hall, N.G., (2007), Supply chain scheduling: Conflict and cooperation in assembly systems. *Operations Research*, 55(6):1072-1089.

[17] Chen, Z.-L., (2010), Integrated production and outbound distribution scheduling: Review and extension. *Operations Research*, 58(1):130-148.

[18] Cheng, T.C.E., Ding, Q. and Lin, B.M.T., (2004), A concise survey of scheduling with time-dependent processing times, *European Journal of Operational Research*, **152**, 1-13.

[19] Cheng, T.C.E., Kang, L., and Ng, C.T., (2004), Due-date assignment and single machine scheduling with deteriorating jobs, *Journal of the Operational Research Society*, **55**, 198-203.

[20] Cook, S.A., (1971), The complexity of theorem-proving procedures. *In Proceedings of the 3rd ACM Symposium on the Theory of Computing*, pages 151-158.

[21] Cook, W.J., Cunningham, W.R., Pulleyblank, W.R. and Schrijver, A., (1998), *Combinatorial Optimization*. J. Wiley.

[22] Dawande, M., Geismar, H.N., Hall, N.G. and Sriskandarajah, C., (2006), Supply chain scheduling: Distribution systems. *Production and Operations Management*, 15(2):243–261.

[23] Deĭneko, V.G. and Filonenko, V.L., (1979). On the Reconstruction of Specially Structures Matrices. *Aktualnyje Problemy EVM i Programmirovanije, Dnepropetrovsk DGU* (in Russian).

[24] Deĭneko, V.G., Shabtay, D. and Steiner, G., (2010), On the asymptotic behavior of subtour-patching heuristics in solving the TSP on permuted Monge matrices, *Journal of Heuristics*, 1-36.

[25] Deĭneko V.G., Steiner, G. and Xue, Z., (1995), Robotic-Cell Scheduling: Special Polynomially Solvable Cases of the Traveling Salesman Problem on Permuted Monge Matrices, *Journal of Combinatorial Optimization*, **9**, 381-399.

[26] Evans, G.W., (1984), An overview of techniques for solving multiobjective mathematical programs. *Management Science*, pages 1268-1282.

[27] Fry, T.D., Armstrong, R.D. and Lewis, H., (1989), A framework for single machine multiple objective sequencing research. *Omega*, 17(6):595-607.

[28] Garey, M.R. and Johnson, D.S., (1979), *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. W.H. Freeman and Co., New York, NY.

[29] Gawiejnowicz, S., (2008), Time-Dependent Scheduling. , Berlin: Springer.

[30] Gordon, V., Proth, J.M. and Chu, C.B., (2002), A Survey of the State-of-the-Art of Common Due Date Assignment and Scheduling Research, *European J. of Oper. Research*, **139**, 1-25.

[31] Gordon, V., Proth, J.M. and Chu, C.B., (2002), Due Date Assignment and Scheduling: SLK, TWK and Other Due Date Assignment Models, *Production Planning and Control*, **13** (2), 117-132.

[32] Gordon, V., Proth, J.M. and Strusevich, V.A., (2004), Scheduling with Due Date Assignment, in Leung, J.Y-T. (ed.), *Handbook of Scheduling,* CRC Press, Boca Raton, FL, 21-1 – 21-22.

[33] Gordon, V., Potts, C.N., Strusevich, V.A. and Whitehead, J.D., (2008), Single machine scheduling models with deterioration and learning: handling precedence constraints via priority generation, *Journal of Scheduling*, **11**, 357-370.

[34] Graham, R.L., Lawler, E.L., Lenstra, J.K. and Rinnooy Kan, A.H.G., (1979), Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey, *Annals of Discrete Mathematics*, **3**, 287-326.

[35] Gupta, J.N.D. and Gupta, S.K., (1988). Single facility scheduling with nonlinear processing times, *Computers and Industrial Engineering*, **14**, 387-393.

[36] Gupta, S.K., Kunnathur, A.S. and Dandapani, K., (1987), Optimal repayment policies for multiple loans. *Omega*, 15(4):323-330.

[37] Hardy, G.H., Littlewood, J.E. and Pólya, G., (1952), *Inequalities*, (2nd ed.), Cambridge Univ. Press, Cambridge.

[38] Hall, N.G. and Potts, C.N., (2003), Supply chain scheduling: Batching and delivery. *Operations Research*, 51(4):566-584.

[39] Hall, N.G. and Potts, C.N., (2005), The coordination of scheduling and batch deliveries. *Annals of Operations Research*, 135(1):41-64.

[40] Hall, N.G., Lei, L. and Pinedo, M., editors. . (2008), *Special Issue on Supply Chain Coordination and Scheduling Annals of Operations Research (161).*

[41] Ham, I., Hitomi, K. and Yashida, T., (1985), *Group technology applications to production management.* Kluwer-Nijhoff Pub.(Boston and Hingham, MA, USA).

[42] Hochbaum, D.S. and Landy, D., (1994), Scheduling with batching: minimizing the weighted number of tardy jobs. *Operations Research Letters*, 16(2):79-86.

[43] Hochbaum, D.S., editor. (1996), *Approximation Algorithms for $\mathcal{NP}$-Hard Problems.* PWS Publishing Company.

[44] Hoogeveen, H., (2005), Multicriteria scheduling. *European Journal of Operational Research*, 167(3):592-623.

[45] Janiak, A. and Rudek, P., (2006). in *Scheduling in Computer and Manufacturing Systems*, Inst. Informat., Automat. and Robotics, Wroclaw, Poland, 26-38.

[46] Janiak, A., Krysiak, T. and Trela, R., (2011). Scheduling problems with learning and ageing effects: a survey. Decision Making in Manufacturing Services 5, 19-36.

[47] Kaminsky, P. and Hochbaum, D., (2004), Due Date Quotation Models and Algorithms, in Leung, J.Y-T. (ed.), *Handbook of Scheduling,* CRC Press, Boca Raton, FL, 20-1 – 20-22.

[48] Koulamas, C., Gupta, S. and Kyparisis, G.J., (2010), A unified analysis for the single-machine scheduling problem with controllable and non-controllable variable job processing times *European Journal of Operational Research* **205** (2), 479-482.

[49] Kovalyov, M.Y., (1995), Improving the complexities of approximation algorithms for optimization problems. *Operations Research Letters*, 17:85-87.

[50] Kunnathur, A.S. and Gupta, S.K., (1990), Minimizing the makespan with late start penalties added to processing times in a single facility scheduling problem. *European Journal of Operational Research*, 47(1):56-64.

[51] Lawler, E.L., Lenstra, J.K. and Rinnooy Kan, A.H.G. and Shmoys, D.B., (1993), Sequencing and scheduling: Algorithms and complexity. *In Handbooks in Operations Research and Management Science, Edited by S.G. Graves et al.*, 4:445-552.

[52] Lee, C.Y. and Uzsoy, R., and Martin-Vega, L.A., (1992), Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, pages 764-775.

[53] Lee, C.Y. and Chen, Z.-L., (2001), Machine scheduling with transportation considerations. *Journal of Scheduling*, 4:3-24.

[54] Lee, W.C., (2004), A Note on Deteriorating Jobs and Learning in Single-Machine Scheduling Problems, *International Journal of Business and Economics* **3** (1), 83-89.

[55] Leyvand, Y., Shabtay, D. and Steiner, G., (2010), A Unified Approach for Scheduling with Convex Resource Consumption Functions Using Positional Penalties, *European Journal of Operational Research* **206**(2), 301-312.

[56] Li, C.L., Vairaktarakis, G. and Lee, C.Y., (2005), Machine scheduling with deliveries to multiple customer locations. *European Journal of Operational Research*, 164:39-51.

[57] Liman, S.D., Panwalkar, S.S. and Thongmee, S., (1998), Common Due Window Size and Location Determination in a Single Machine Scheduling Problem, *J. of the Oper. Research Society*, **49**, 1007-1010.

[58] Manoj, U.V., Gupta, J.N.D., Gupta, S.K. and Sriskandarajah, C., (2008), Supply chain scheduling: Just-in-time environment. *Annals of Operations Research*, 161:53-86.

[59] Mosheiov, G., (1992), V-Shaped policies for scheduling deteriorating jobs, *Operations Research*, **39**(6), 979-991.

[60] Mosheiov, G., (1996), $\wedge$-Shaped policies for scheduling deteriorating jobs, *Journal of the Operational Research Society*, **47**, 1184-1191.

[61] Mosheiov, G., (2001), Scheduling problems with a learning effect, *European Journal of Operational Research*, **132**, 687-693.

[62] Mosheiov, G., (2001), Parallel machine scheduling with a learning effect, *Journal of the Operational Research Society*, **52**(10), 1165-1169.

[63] Mosheiov, G. and Sidney, J.B., (2005), Note on scheduling with general learning curves to minimize the number of tardy jobs, *Journal of the Operational Research Society*, **56**, 110-112.

[64] Mosheiov, G, (2005). A note on scheduling deteriorating jobs. *Mathematical and Computer Modelling*, 41, 883-886.

[65] Nagar, A., Haddock, J. and Heragu, S., (1995). Multiple and bicriteria scheduling: A literature survey. *European journal of operational research*, 81(1), 88-104.

[66] Ng, C.T., Cheng, T.C.E. and Bachman, A., (2002). Three scheduling problems with deteriorating jobs to minimize the total completion time. *Information Processing Letters*, **81**(6), 327-333.

[67] Panwalkar, S.S., Smith, M.L. and Seidmann, A., (1982), Common Due Date Assignment to Minimize Total Penalty for the One Machine Scheduling Problem, *Oper. Research*, **30**, 391-399.

[68] Papadimitriou, C.H., (1994), *Computational Complexity.* Addison-Wesley, Reading, MA.

[69] Papadimitriou, C.H. and Steiglitz, K., (1982), *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ.

[70] Pinedo, M., (2001), *Scheduling: Theory, Algorithms, and Systems.* Prentice-Hall, 2nd edition.

[71] Potts, C.N. and Kovalyov, Y.M., (1980) Analysis of a heuristic for one machine sequencing with release dates and delivery times. *Operations Research*, 28:1436-1441.

[72] Potts, C.N. and Kovalyov, Y.M., (2000), Scheduling with batching: A review. *European Journal of Operational Research*, 120:228-249.

[73] Pundoor, G. and Chen, Z.-L., (2005), Scheduling a production-distribution system to optimize the tradeoff between tardiness and total distribution cost. *Naval Research Logistics*, 52:571-589.

[74] Rachaniotis, N.P. and Pappis, C.P., (2006). Scheduling fire-fighting tasks using the concept of "deteriorating jobs". *Canadian Journal of Forest Research*, 30(3), 652-658.

[75] Rustogi, K. and Strusevich, V.A., (2012). Simple Matching vs Linear Assignment in Scheduling Models with Positional E¡èects: A Critical Review. *European Journal of Operational Research*, 222, 393-407.

[76] Rustogi, K. and Strusevich, V.A., (2012). Single machine scheduling with general positional deterioration and rate-modifying maintenance. *European Journal of Operational Research*, 40(6), 791-804.

[77] Sáenz-Royo, C., and Salas-Fumás, V., (2012), Learning to learn and productivity growth: Evidence from a new car-assembly plant, *Omega*, http://www.sciencedirect.com/science/article/pii/S0305048312000849.

[78] Seidmann, A., Panwalkar, S.S. and Smith, M.L., (1981), Optimal Assignment of Due Dates for a Single Processor Scheduling Problem, *International J. of Production Research*, **19**, 393-399.

[79] Selvarajah, E. and Steiner, G., (2006) Batch scheduling in a two-level supply chain - a focus on the supplier. *European Journal of Operational Research*, 173(1):226-240.

[80] Selvarajah, E. and Steiner, G., (2006), Batch scheduling in customer-centric supply chains. *Journal of the Operations Research Society of Japan*, 49(3):174-187.

[81] Selvarajah, E. and Steiner, G., (2009), Approximation algorithms for the supplier's supply chain scheduling problem to minimize delivery and inventory holding costs. *Operations Research*, 5(2):426-438.

[82] Shabtay, D. and Steiner, G., (2006), Two Due Date Assignment Problems in Scheduling a Single Machine, *Oper. Research Letters,* **34**, 683-691.

[83] Shabtay, D. and Steiner, G., (2007), Optimal Due Date Assignment and Resource Allocation to Minimize the Weighted Number of Tardy Jobs on a Single Machine, *Manufacturing & Service Operations Management*, **9**(3), 332-350.

[84] Slotnick, S.A. and Sobel, M.J., (2005), Manufacturing lead-time rules: Customer retention versus tardiness costs, *European J. of Oper. Research*, **169**, 825-856.

[85] Steiner, G. and Zhang, R., (2009), Approximation algorithms for minimizing the total weighted number of late jobs with late deliveries in two-level supply chains. *Journal of Scheduling*, 12(6):565-574.

[86] Steiner, G. and Zhang, R., (2011), Minimizing the weighted number of tardy jobs with due date assignment and capacity-constrained deliveries. *Annals of Operations Research*, 191(1): 171-181.

[87] Tang, J., Yung, K.-L., KaKu, I. and Yang, J., (2008), The scheduling of deliveries in a production-distribution system with multiple buyers. *Annals of Operations Research*, 161:5-23.

[88] Thomas, D.J. and Griffin, P.M., (1996), Coordinated supply chain management. *European Journal of Operational Research*, 94:1-15.

[89] Vazirani, V.V., (2003), *Approximation Algorithms*. Springer, Berlin, German.

[90] Wang, J.B., (2006), A note on scheduling problems with learning effects and deteriorating jobs, *International Journal of Systems Science*, **37**, 827-833.

[91] Wang, L.Y., Wang, J.B., Guo, W.J., Huang, X. and Feng, E.M., (2007), Two single-machine scheduling problems with the effects of deterioration and learning, *International Journal of Advanced Manufacturing Technology*, **46**, 715C720.

[92] Wang, J.B., (2007), Single-machine scheduling problems with the effects of learning and deterioration, *Omega*, **35**, 397-402.

[93] Wang, J.B. and Cheng, T.C.E, (2007), Scheduling problems with the effects of learning and deterioration, *Asia-Pacific Journal of Operational Research*, **24**(2), 245-261.

[94] Wang, X.L. and Cheng, T.C.E, (2007), Single-machine scheduling with deteriorating jobs and learning effects to minimize the makespan, *European Journal of Operational Research*, **178**, 57-70.

[95] Webb, G.K., (1994), Integrated circuit (IC) pricing, *High Technology Management Research*, **5**(2), 247-260.

[96] Williams, J.F., (1981), A hybrid algorithm for simultaneous scheduling of production and distribution in multi-echelon structures. *Management Science*, 29:77-92.

[97] Woeginger, G.J., (2008), Open problems around exact algorithms. *Discrete Applied Mathematics*, 156:397-405.

[98] Wright, T.P., (1936), Factors affecting the cost of airplanes, *Journal of Aeronautical Sciences*, **3**, 122-128.

[99] Yang, X.G., (2000), Scheduling with generalized batch delivery dates and earliness penalties. *IIE Transactions*, 32:735-741.

[100] Yang, D. L. and Kuo, W. H., (2009). Single-machine scheduling with both deterioration and learning e¡èects. *Annals of Operational Research*, 172, 315-327.

Appendix A

Notations

- $J = \{1, ...n\}$: the set of jobs;
- p_j : the "normal" processing time of job j;
- $a_{[j]}$: the actual processing time of the job at the j-th position;
- c: the rate of learning;
- b: the rate of deteriorating;
- $S_{[j]}$: the starting time of the job at the j-th position
- d_j : the due date of job j;
- $[\underline{d}, \overline{d}]$: assigned due date time window in the CONW method;
- w_j : the tardiness penalty (weight) of job j;
- D_j : the assigned due date of job j in the DIF problems, which is a decision variable;
- ξ_i: a *positional, job-independent penalty* for *any* job scheduled in the ith position;
- $\eta_{[j]}$: a certain parameter of job j
- α : the cost of one unit of earliness;
- β : the cost of one unit of tardiness;
- γ : the cost of one unit of due date and operation time;
- A_j : the contracted due date of job j;

- A : the common contracted due date, i.e., $A_j = A$, $\forall j \in J$;
- $R_j = \max\{D_j - A_j, 0\}$: the extended time units by D_j on A_j;
- λ_j : the due-date-assignment cost per extended time unit of R_j;
- λ : the uniform due-date-assignment cost per extended time unit of R_j, i.e., $\alpha_j = \alpha$, $\forall j \in J$;
- $\lambda_j R_j$: the due-date-assignment cost of job j;
- C_j : the completion time of job j, which is defined by the batch-completion time of the same batch;
- U_j : the tardiness indicator of job j: $U_j = 1$, if job j is tardy and $U_j = 0$ otherwise;
- $C(i)$: the batch-completion time of batch i;
- $d(i)$: the batch-due date of batch i;
- Δ : the batch-setup time before processing the first job in each batch;
- z : the number of batches in a schedule;
- T : the minimum time between any two consecutive deliveries;
- ϕ : the maximum number of jobs in a single batch (delivery);