SUPPLY CHAIN SCHEDULING WITH DELIVERY COSTS

SUPPLY CHAIN SCHEDULING WITH DELIVERY COSTS

By

RUI ZHANG, B.ENG., B.SC., M.SC.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Doctor of Philosophy

McMaster University

DOCTOR OF PHILOSOPHY (2010)	McMaster University
(BUSINESS ADMINISTRATION)	Hamilton, Ontario

TITLE:	Supply chain scheduling with delivery costs
AUTHOR:	Rui Zhang
	B.Eng (Qingdao University, P.R. China)
	B.Sc (McMaster University, Canada)
	M.Sc (McMaster University, Canada)
SUPERVISOR:	Dr. George Steiner, Professor
NUMBER OF PAGES:	xiii, 140

To my wife, miao and my daughter, zoey

Abstract

Supply chain management has been one of the most important issues in manufacturing industries. In order to improve customer satisfaction, suppliers usually extend their inbound production operations to outbound logistics operations. To improve the overall operational performance, it is necessary to study scheduling models which consider inbound production and outbound deliveries simultaneously. This thesis deals with supply chain problems on the operational level using deterministic models.

Meeting due dates is always one of the most important concerns in scheduling and supply chain management. In most supplying contracts, customers require that suppliers either meet contracted due dates or pay tardiness penalties. In order to save delivery costs, suppliers usually deliver jobs in batches. Therefore, we will study supply chain scheduling problems with delivery costs, where our goal is to minimize the sum of the weighted number of tardy jobs and the batch-delivery costs on a single machine.

In traditional manufacturing system, due dates are not considered as given by exogenous decisions. In modern supply chains, however, due dates are determined by taking into account the system's ability to meet the assigned due dates, which can be quoted with certain costs. Therefore, we will study supply chain scheduling problems with delivery costs and due date assignment, where our goal is to minimize the sum of the weighted number of tardy jobs, the due-date-assignment costs and the batch-delivery costs on a single machine.

As we know, most machine scheduling models are intractable in terms of computational complexity. Therefore, for our above problems, which are even harder, we first prove their computational complexity. Then we propose pseudo-polynomial algorithms for optimal solutions. For some problems, the pseudo-polynomial algorithms perform in polynomial time for some special cases. Finally, we develop efficient approximation algorithms or fully polynomial time approximation schemes, which can be implemented easily in practice.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Professor George Steiner, for providing me the opportunity to be his student, for his financial support, and for his continued encouragement in the past four years. In particular, I would like to thank him for his supervision from the initial selection of a research topic until the final writing of this thesis.

I would also like to thank Professor George Wesolowsky and Professor Elkafi Hassini for their constructive comments and valuable suggestions which improved the quality of this thesis. I would like to thank my external examiner Professor Nicholas Hall for providing helpful remarks on this thesis. I would like to express my thanks to Professor Prakash Abad for his encouragement and support throughout my studies. I also cannot forget the great time with Tiina Salisbury when I was the teaching assistant for her courses. I thank all my colleagues who made my time at DeGroote School of Business enjoyable. I thank all the administration staff who provided kindness and support during my studying.

I would like to thank Mr. Ni Jin, Mrs. Lu Mei and their son Nathan for their help after we arrived in this country.

I am deeply indebted to my parents for their love and for giving me life. I am deeply thankful to my wife Miao for her love and support and for always being there for me. I would like to give a lot of thanks to my newborn daughter Zoey for the happiness that she has brought to me with her birth.

Contents

List of Figures

Chapter 1

Introduction

Scheduling is concerned with allocating scarce resources to tasks over time. It plays a crucial role in manufacturing and service industries. Typically, in a scheduling problem, it is required to determine a schedule, which achieves certain objective(s) without violating any accompanying constraints. Scheduling problems that we will discuss in this thesis are categorized as *deterministic off-line scheduling*, where all data are 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 1970's. It involves assigning limited resources (single or multiple machines) to a set of tasks (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 by Brucker [2001] and Pinedo [2001].

1.1.1 Machine Scheduling Models

In this section, we will introduce the classical notation system for machine scheduling models and relevant concepts that will be used later. In machine scheduling, models can be classified by the configuration of machines, the nature of jobs and the objective. In the *default setting*, a machine can only execute one job at a time and a 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.

In what follows, we outline the *three-field notation* system, $\alpha|\beta|\gamma$, which was established by Graham et al. [1979] for representing 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 involves the type of machines. For instance, $\alpha = P2$ implies a *parallel machine* environment consisting of two machines;

- β indicates the nature of jobs, i.e., the restrictions and the constraints of processing a job. For instance, $\beta = pmtn$ implies that *preemption* is allowed such that a job can be interrupted during its processing and started over sometime later. In particular, if β is left blank, this denotes the default setting;

- 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.

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 by Lawler et al. [1993] and Chen et al. [1998].

1.1.2 Algorithms in Machine Scheduling

The term *algorithm* refers to a series of instructions for solving a given problem. The fundamental issue of an algorithm is the *efficiency* for finding the best 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 concepts are central in *computational complexity theory*, which is developed to study the nature of algorithmic tractability of problems and is widely used in computer science and combinatorial optimization. In computational complexity theory, one of the major concerns about a problem is if it is $\mathcal{NP}$-hard or not. Here the term "$\mathcal{NP}$" stands for "*nondeterministic polynomial time*". A *decision problem* is in the class $\mathcal{NP}$, if its "Yes" 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 Cook [1971] is the *satisfiability problem*, known as SAT: Is there a truth assignment for a given boolean formula? In his famous paper, he proved SAT is $\mathcal{NP}$-hard. This proof 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 $\mathcal{NP}$ and then a known $\mathcal{NP}$-complete problem has to be reducible to the problem in polynomial time. This proof strategy will be widely used in this thesis. 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 an-

nounced by the Clay Mathematics Institute of Cambridge, Massachusetts (CMI) in 2000 - http://www.claymath.org/millennium/P_vs_NP/ (July 29, 2008). In general, we believe $\mathcal{P} \neq \mathcal{NP}$ [Garey and Johnson, 1979], [Papadimitriou, 1994], [Hochbaum, 1996], [Vazirani, 2003], [Cook et al., 1998].

We outline the categorizations of algorithms used in machine scheduling. From the view of efficiency, algorithms fall into the following three categories: *polynomial*, *pseudo-polynomial* and *nonefficient*. Before giving the definitions, we introduce the "Big-O" notation. A function $T(n)$ is $O(f(n))$, if there exists a constant c and a number X such that for all $n \geq X$, it is always true that $T(n) \leq cf(n)$. For a problem, an algorithm is called polynomial if its running time $T(n) = O(n^k)$, where k is constant and the size of the problem is polynomial 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 constant and $O(N)$ is the size of the problem under unary encoding. Any algorithm with higher running time is called nonefficient, e.g., an algorithm with running time $T(n) = O(2^n)$ and the size of problem under binary encoding. The most recent developments on nonefficient algorithms in combinatorial optimization are reviewed by Woeginger [2008].

Based on computational complexity theory, an $\mathcal{NP}$-hard problem is called $\mathcal{NP}$-hard only in the *ordinary* sense, if 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 approximation 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 and $1/\varepsilon$, where ε could be any given positive value. 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 by Hochbaum [1996] and Vazirani [2003].

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*" [Cook et al., 1998], 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 Supply Chain Scheduling Models with Delivery Costs

Supply chain management has been one of the most important topics in manufacturing research. Most of the supply chain literature focuses on issues on the strategic level, using stochastic models. By the survey paper [Thomas and Griffin, 1996], how-

ever, 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. 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 [2003].

In contrast with classical machine scheduling, in supply chain scheduling, deliveries are also 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 that consumes machine time, the delivery operation does bring a new question: *How to group jobs in batches for both production and deliveries*? This makes supply chain scheduling connected to *batch scheduling*, which is a well-studied research area in machine scheduling. They are different, however, as delivery costs are not part of objectives in batch scheduling. Detailed discussions about batch scheduling can be found in the survey paper by Potts and Kovalyov [2000]. 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 [Pinedo, 2001], but its supply chain scheduling version of *minimizing the sum of the total weighted completion times and the batch-delivery costs on a single machine* is strongly $\mathcal{NP}$-hard [Hall and Potts, 2003].

Extending the β field, the three-field notation system can still be applicable in supply chain scheduling. For instance, $1|s|\sum w_jU_j + bq$ 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 s is the machine time needed to set up a new batch, q is the delivery cost per batch and b is the number of batches.

1.3 Models with Delivery Costs and Due Date Assignment

Meeting due dates is one of the most important objectives in scheduling and supply chain management. Customers require that suppliers either meet *contracted due dates* or pay large penalties. For example, Slotnick and Sobel [2005] mentioned that the tardiness penalties in aerospace industries may be as high as one million dollars per day for suppliers of aircraft components. In traditional machine scheduling models, due dates are considered as given by exogenous decisions [Baker and Scudder, 1990]. In an integrated system, in order to avoid tardiness penalties, due dates are determined by taking into account the system's ability to meet the *assigned due dates*. Therefore, suppliers are under increasing pressure to quote attainable due dates. It is obvious that if all due dates are extended to be large enough, then no job would be tardy. In the meantime, extending due dates too far into the future may force a supplier to offer price discounts in order to retain the customer. Thus for any job, there is a trade-off: paying penalty for it being tardy or paying a certain cost to extend the due date. This caused an increasingly large number of recent scheduling studies to take into account due date assignment. And the studies showed that the ability of controlling due dates can be a major factor in improving supply chain performance.

Without including delivery costs, this type of scheduling was named *scheduling with due date assignment*. A large number of publications in this research area can be found in the survey papers by Gordon et al. [2002a] and Gordon et al. [2002b]. In their papers, the following notations are used for different types of assignable due dates: CON means that there will be a *common* due date D assigned to all jobs and DIF means that *distinct* due dates D_j, $j = 1, ..., n$, can be assigned to each job individually. When all assigned due dates are *a function of the processing time*, we consider two cases: SLK means that the assigned due dates are the sum of the processing time and a non-negative *slack*, i.e., $D_j = p_j + \theta$, $\theta \geq 0$ and TWK means that the assigned due dates are the product of the processing time and a non-negative

coefficient, i.e., $D_j = \eta p_j$, $\eta \geq 0$. For the most recent developments in this research area, we refer to the survey papers by Kaminsky and Hochbaum [2004] and Gordon et al. [2004].

Including the above notations in the β field, the three-field notation system is still applicable for supply chain scheduling models with delivery costs and due date assignment. For instance, $1|s,\text{DIF}|\sum \alpha_j \max\{D_j - A_j, 0\} + \sum w_j U_j + bq$, where A_j is the original contracted due date of job j and α_j is the *due-date-assignment cost per extended time unit* from A_j to D_j, denotes a supply chain scheduling model of minimizing the sum of the weighed number of tardy jobs, the due-date-assignment costs and the batch-delivery costs on a single machine for a single customer. Recall that s is the machine time needed to set up a new batch, q is the delivery cost per batch and b is the number of batches.

Chapter 2

Literature Review

This thesis is mainly about *minimizing the weighted number of tardy jobs on a single machine*, denoted by $1||\sum w_jU_j$, in the supply chain context. In this chapter, we will review the relevant scheduling literature.

2.1 Machine Scheduling Models

Moore [1968] found a dynamic programming algorithm to minimize the number of tardy jobs, denoted by $1||\sum U_j$, in $O(n \log n)$ time, where n is the number of jobs. Karp [1972] proved that the weighted problem, $1||\sum w_jU_j$, is $\mathcal{NP}$-hard. Four years later, the equivalent maximization version of the $1||\sum w_jU_j$ problem (*maximizing the weighted number of on-time jobs*) was shown to be solvable in pseudo-polynomial time by Sahni [1976]. This established that the $1||\sum w_jU_j$ problem is $\mathcal{NP}$-hard only in the ordinary sense. Moreover, this pseudo-polynomial algorithm was further converted into an FPTAS in the same paper [Sahni, 1976] with time complexity $O(n^3/\varepsilon)$. Gens and Levner [1979b] presented an FPTAS for the original minimization version of the $1||\sum w_jU_j$ problem with the same time complexity $O(n^3/\varepsilon)$. By adding a special binary search into the FPTAS in [Gens and Levner, 1979b], Gens and Levner

[1981] were able to improve the time complexity to $O(n^2/\varepsilon + n^2 \log n)$. As far as we know, this is the best FPTAS for the $1||\sum w_j U_j$ problem in terms of time complexity. By allowing batching in the $1||\sum w_j U_j$ problem, Hochbaum and Landy [1994] studied the $1|s|\sum w_j U_j$ problem, where s is the machine time needed to set up a new batch, and found a backward dynamic programming algorithm for it, which runs in pseudo-polynomial time and shows that the $1|s|\sum w_j U_j$ problem is $\mathcal{NP}$-hard only in the ordinary sense. For the same problem, Brucker and Kovalyov [1996] found a forward dynamic programming algorithm running in pseudo-polynomial time and then converted the forward dynamic programming algorithm into an FPTAS with time complexity $O(n^3/\varepsilon + n^3 \log n)$.

Many other problems related to the $1||\sum w_j U_j$ problem and with further constraints were studied in the past three decades. Lawler [1983] proved that the $1|\bar{d}_j|\sum w_j U_j$ problem is $\mathcal{NP}$-hard, where job j has to be completed by the *deadline*, $\bar{d}_j$, in any feasible schedule. This problem, however, is still open whether it is solvable by a pseudo-polynomial algorithm. Lenstra et al. [1977] provided a strong $\mathcal{NP}$-hardness proof for the $1|r_j|\sum U_j$ problem, where job j can not be executed until the *release time*, r_j, in any feasible schedule. Bar-Noy et al. [2001] studied the weighted $1|r_j|\sum w_j U_j$ problem. Lawler [1990] presented a pseudo-polynomial algorithm for the $1|r_j, pmtn|\sum w_j U_j$ problem and the algorithm runs in $O(n^5)$ time when all jobs have unit weights, i.e., $w_j = 1$, denoted by $1|r_j, pmtn|\sum U_j$. Baptiste [1999] improved the complexity to $O(n^4)$. Other related problems can be found in the survey paper by Akker and Hoogeveen [2004].

2.2 Supply Chain Scheduling Models with Delivery Costs

One of the first scheduling models taking into account logistics was by Potts and Kovalyov [1980]. However, their model considers only delivery time but does not

consider delivery costs. Cheng and Kahlbacher [1993] were the first to study a machine scheduling model involving delivery costs. Lee and Chen [2001] further extended the model in [Cheng and Kahlbacher, 1993] into one with a limited delivery capacity.

Hall and Potts [2003] were the first to introduce delivery costs into the objective of 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 even when jobs are for a fixed number of multiple customers. The algorithm becomes polynomial, when all jobs have equal weights. Their model, however, does not consider batch-setup time, which consumes machine time in most real cases.

In the same paper [Hall and Potts, 2003], they also studied other objectives of minimizing the overall scheduling and delivery costs. As it is a new area for research, there are relatively few papers dealing specifically with scheduling problems in supply chains. Chen and Hall [2007] extended these to supply chains with assembly-type manufacturing systems and Dawande et al. [2006] 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 papers by Williams [1981], and Lee and Chen [2001], which consider the integration of transportation time and capacity issues with scheduling decisions. Li et al. [2005] studied the problem of minimizing the average job-arrival times, which include travel times to the customers. Chen and Vairaktarakis [2005] presented polynomial time solutions (with a fixed number of customers) 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. Pundoor and Chen [2005] studied a model where the objective is to minimize a convex combination of the maximum delivery tardiness and total delivery costs. Selvarajah and Steiner [2006a,b, 2009] developed exact and approximation algorithms for the supplier's problem of minimizing the sum of the total weighted flow time and batch-delivery costs. Agnetis et al. [2006] looked at the problem of rescheduling to resolve conflicts between the

supplier's and the manufacturers' ideal schedules. Hall and Potts [2005] studied the coordination of scheduling and batch deliveries with various scheduling objectives. Moreover, Hall et al. [2008] edited a special issue focusing on the area of supply chain coordination and scheduling, in which Tang et al. [2008] considered a production and distribution model taking into account inventory control issue with one supplier and multiple buyers. Manoj et al. [2008] studied decentralized and joint optimization results of a model with a manufacturer, a distributor and several retailers in a just-in-time environment. The most recent comprehensive survey paper was by Chen [2008].

2.3 Models with Delivery Costs and Due Date Assignment

Cheng and Kovalyov [1996] studied the first batch scheduling model which took into account due-date-assignment costs: minimizing 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 the pseudo-polynomial algorithm 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 [2006]. In their model, each job has a contracted due date, and each job can be assigned an arbitrary 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 provided a strong $\mathcal{NP}$-hardness proof for the general case and then presented two polynomial algorithms for two special cases: one with equal due-date-assignment costs and zero contracted due dates and

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.

There is some literature which considers optimizing other scheduling measures in this context. Chen [1996] 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 [2000] focused only on tardiness penalties with quoted delivery dates but without delivery costs. In this paper, two problems are studied: one to minimize the total batch earliness and the other one to minimize the largest batch earliness.

2.4 Outline

The thesis is organized as follows, in Chapter 3 we introduce some terminologies, assumptions and notations that will be used in later Chapters $4-8$, and we also present a bound improvement procedure which will be used in Chapters $4-7$. In Chapters 4 and 5, we study supply chain scheduling models with delivery costs for a single customer and multiple customers, respectively. In Chapters 6, 7 and 8, we study supply chain scheduling models with both delivery costs and due date assignment with respect to different types of assigned due dates: Chapter 6 deals with models with a common due date which is assigned to all jobs; Chapter 7 deals with models with arbitrary due dates which are assigned to each job individually; and Chapter 8 deals with models with assigned due dates which are based on the processing times. Chapter 9 includes our final conclusions and discusses future research potential.

Chapter 3

Preliminaries

In this chapter, we first describe some terminologies, establish some fundamental assumptions and introduce some notations for models which will be used in this thesis. (Notice that these terminologies, assumptions and notations are listed in Appendixes A, B and C as well.) Then we propose a bound improvement procedure which will be used in Chapters 4, 5, 6 and 7.

3.1 Terminologies

As mentioned previously, supply chain scheduling has a strong connection with batch scheduling where jobs are processed in batches. In contrast with traditional machine scheduling, the completion time of a job is defined by the completion time of the last job in the same batch. We call this the *batch-completion time*. Similarly, we call the smallest due date of jobs in the same batch the *batch-due date*.

Suppose batch i has a batch-completion time $C(i)$ and a batch-due date $d(i)$. Let $J(i)$ be the job set scheduled in batch i. If $C(i) \leq d(i)$, then all jobs in batch i are early. Because the jobs in $J(i)$ have a common completion time $C(i)$ and their due date is at least $d(i)$, these jobs are early. Then we call batch i an *early batch*.

Let job j be the one which has the largest due date in $J(i)$. If $C(i) > d_j$ then the jobs in $J(i)$ complete later than their due dates, and these jobs are tardy. Then we call batch i a *tardy batch*. If $d(i) < C(i) \leq d_j$, then there is at least one early job j with the due date d_j and one tardy job with the due date $d(i)$ in $J(i)$. Then we call batch i a *mixed batch*.

3.2 Assumptions

In supply chain scheduling, a job has to go through two stages: the production and the delivery operations. Suppose at time t_j, job j completes the production operation and is ready to be delivered. Since delivering jobs in batches saves costs, the supplier may want to deliver job j and other jobs in a single batch, say batch i. Let job k be the latest finished job in batch i. Suppose job k completes the production operation at time t_k, then job j can be delivered at t_k together with other jobs in batch i. Therefore, the batch-completion time of batch i is defined by the time when all the jobs in batch i are ready to be delivered, i.e., $C(i) = t_k > t_j$. This is also the completion time of job j and k, i.e., $C_j = C_k = C(i)$.

Suppose it takes time τ for a vehicle to deliver a batch to a customer. Then finally, at time $t_k + \tau$, job j is available to the customer. Similarly to definitions in machine scheduling, we use the tardiness indicator U_j for job j in the supply chain context, i.e., $U_j = 1$, if $C_j + \tau = t_k + \tau > d_j$ and $U_j = 0$, otherwise. It is not hard to see that replacing d_j by $d_j - \tau$ and setting $\tau = 0$ would not change the value of tardiness indicator U_j. Therefore, if we assume that the number of vehicles available for delivering jobs is unlimited (*unlimited-delivery*), then we are able to deliver a batch any time. With the unlimited-delivery assumption, we are able to assume $\tau = 0$ (*instant-delivery*). Notice that once all due dates are reduced by τ, we reset $\tau = 0$. Other than these, we have two more assumptions about deliveries: that all jobs have to be delivered to customers including tardy jobs (*tardy-delivery*), and that only jobs for the same customer can be delivered in a batch (*batch-delivery*).

For inbound production operations, we assume that there is only one machine for processing jobs (*single-machine*). Moreover, we assume that all jobs are available for processing at time zero (*zero-availability*) and no interruption is allowed during jobs' processing (*non-preemption*).

3.3 Notations

In Chapters 4, 6, 7 and 8, we are given a job set $J = \{1, 2, ..., n\}$ for a single customer. For each job $j \in J$, we use notation p_j for the *processing time*, d_j for the *due date* and w_j for the *tardiness penalty* of job j, $\forall j \in J$. Let s be the *batch-setup time* before processing the first job in each batch and q be the *batch-delivery cost* of each batch. Let C_j be completion time of job j. If $C_j > d_j$, then the tardiness indicator $U_j = 1$. Otherwise, $U_j = 0$. In denoting a scheduling problem, we use notation b to represent the *number of batches* in a schedule, i.e., $1||\sum w_j U_j + bq$.

In Chapters 6, 7 and 8, we use notation $A_j \geq 0$ 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*, which is used to determine the tardiness indicator U_j such that: if $C_j > D_j$, then $U_j = 1$; if $C_j \leq D_j$, then $U_j = 0$. Let $R_j = \max\{D_j - A_j, 0\}$ be the *extended time units* on A_j by D_j and $\alpha_j R_j$ be the *due-date-assignment cost*, where α_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., $A_j = A$ and if all D_j are equal, then we use notation D to represent the *common assigned due date*, i.e., $D_j = D$.

In Chapter 5, we are given a set of customers, $M = \{1, ..., m\}$ and a set of job $J = \{J_1, ..., J_m\}$, where $J_i = \{(i, 1), ..., (i, n_i)\}$, $\forall i \in M$ and $n = \sum_{i=1}^{m} n_i$. For each job $(i, k) \in J$, we have $p_{(i,k)}$ (*processing time*), $d_{(i,k)}$ (*due date*), $w_{(i,k)}$ (*tardiness penalty*) and $U_{(i,k)}$ (*tardiness indicator*). For each customer $i \in M$, we have s_i (*batch-setup time*), q_i (*batch-delivery cost*) and b_i (*number of batches*).

3.4 Bound Improvement Procedure

Bound improvement procedure was first introduced by Chudanov et al. [2006]. In this thesis, we denote it by $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$. Assume that x^* is the optimal solution value for a minimization problem $\mathcal{P}$. The procedure requires two parameters as the inputs: (1) positive values a_1 and a_2, which indicate an interval such that $x^* \in [a_1, a_2]$; (2) an algorithm $\mathcal{A}(\omega, \mathcal{P})$, which for any given ω runs on the problem $\mathcal{P}$, and reports either a value $x^* > 2\omega/3$, or a feasible solution with value $x \leq \omega$ in $O(\pi)$ time, where π is a polynomial function of the input size for the problem $\mathcal{P}$. The output of the procedure is a positive value ξ, which indicates an interval such that $x^* \in [\xi, 3\xi]$. In Chapter 4, 5, 6 and 7, we call $[a_1, a_2]$ initial bounds and $[\xi, 3\xi]$ tight bounds. In this thesis, we always have $a_2 = na_1$.

$\underline{\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]}$

1. Set $\xi = 2^{\lceil \log(a_2/a_1) \rceil} a_1/3$, $l_1 = 0$ and $l_2 = \lceil \log(a_2/a_1) \rceil$.
2. Set $k = \lceil (l_1 + l_2)/2 \rceil$, $\omega = 2^{k-1} a_1$.
3. Run $\mathcal{A}(\omega, \mathcal{P})$ on the problem $\mathcal{P}$:
   (a) If $\mathcal{A}(\omega, \mathcal{P})$ reports $x^* > 2\omega/3$, then: if $l_2 = k$, then stop; otherwise set $l_1 = k$ and go to step 2 /*l_1 *gets updated only in this step.*
   (b) If $\mathcal{A}(\omega, \mathcal{P})$ reports a solution with value $x \leq \omega$, then: if $l_2 = k$, then set $\xi = \omega/3$ and stop; otherwise set $l_2 = k$, $\xi = \omega/3$ and go to step 2/*l_2 *gets updated only in this step.*

Theorem 3.4.1 *$\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ runs in $O(\pi \log\log(a_2/a_1))$ time and determines a bound for x^* such that $x^* \in [\xi, 3\xi]$.*

Proof. $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ does binary search on $[0, \lceil \log(a_2/a_1) \rceil]$, thus $\mathcal{A}(\omega, \mathcal{P})$ will be called $O(\log\log(a_2/a_1))$ times. If $\mathcal{A}(\omega, \mathcal{P})$ runs in $O(\pi)$ time, then the overall running time is $O(\pi \log\log(a_2/a_1))$. We prove the correctness by studying the following four cases.

(1) Suppose that $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ stops without reporting $x^* > 2\omega/3$ during the whole procedure. In this situation, since always l_2 is reduced in step $3(b)$, $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ must end with $l_1 = 0$ and $l_2 = 1$. This implies that $\mathcal{A}(\omega, \mathcal{P})$ has $\omega = a_1$ in its last run. Setting $\xi = \omega/3 = a_1/3$ implies that $x \leq \omega = 3\xi = a_1$. Because $x^* \in [a_1, a_2]$ and $x^* \leq x$, so we must have $x^* = a_1 \in [\xi, 3\xi]$.

(2) Suppose that $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ stops without ever finding a solution with value $x \leq \omega$ during the whole procedure. In this situation, since always l_1 is increased in step $3(a)$, $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ must end with $l_1 = \lceil \log(a_2/a_1) \rceil - 1$ and $l_2 = \lceil \log(a_2/a_1) \rceil$. This implies that in its last run $\mathcal{A}(\omega, \mathcal{P})$ has $k = 2\lceil \log(a_2/a_1) \rceil$ and $\omega = 2^{\lceil \log(a_2/a_1) \rceil - 1} a_1$. Because ξ has been set as $2^{\lceil \log(a_2/a_1) \rceil} a_1/3$ in step 1 and has never been reset, $x^* > 2\omega/3 = 2^{\lceil \log(a_2/a_1) \rceil} a_1/3$, which implies $x^* > \xi$. Because $x^* \in [a_1, a_2]$, we have $x^* \leq a_2 \leq 2^{\lceil \log(a_2/a_1) \rceil} a_1 = 3\xi$. Therefore, $x^* \in [\xi, 3\xi]$.

(3) Suppose that at some iteration when $k = k_1$, $\mathcal{A}(\omega, \mathcal{P})$ reports $x^* > 2\omega^{(k_1)}/3 = 2^{k_1} a_1/3$ and after this, $\mathcal{A}(\omega, \mathcal{P})$ continuously reports solutions with value $x \leq \omega$ and finally $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ stops with a solution value $x \leq \omega^{(k_2)} = 2^{k_2 - 1} a_1$ at some iteration when $k = k_2$. In this situation, we know that $l_1 = k_1$, $l_2 = k_2 = l_1 + 1$ and $\xi = 2^{k_2 - 1} a_1/3$. Because $\xi = 2^{k_2 - 1} a_1/3 = 2^{k_1} a_1/3 \leq x^* \leq 2^{k_2 - 1} a_1 = 3\xi$, we have $x^* \in [\xi, 3\xi]$.

(4) Suppose that at some iteration when $k = k_1$, $\mathcal{A}(\omega, \mathcal{P})$ reports a solution with value $x \leq \omega^{(k_1)} = 2^{k_1 - 1} a_1$ and after this, $\mathcal{A}(\omega, \mathcal{P})$ continuously reports $x^* > 2\omega/3$ and finally $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ stops with $x^* > 2\omega^{(k_2)}/3 = 2^{k_2} a_1/3$ at some iteration $k = k_2$. In this situation, we know that $l_1 = k_2$, $l_2 = k_1 = l_1 + 1$ and $\xi = \omega^{(k_1)}/3 = 2^{k_2} a_1/3$. Because $\xi = 2^{k_2} a_1/3 \leq x^* \leq 2^{k_1 - 1} a_1 = 2^{k_2} a_1 = 3\xi$, we have $x \in [\xi, 3\xi]$.

In conclusion, $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ determines the interval $[\xi, 3\xi]$ so that $x^* \in [\xi, 3\xi]$. ■

Chapter 4

Single-customer Model with Delivery Costs

In this chapter, we study a supply chain scheduling problem with delivery costs for a single customer.

4.1 Introduction

Hall and Potts [2003] were the first to introduce batch-delivery costs into the objective of minimizing the weighted number of tardy jobs on a single machine, denoted by $1||\sum w_j U_j + bq$. In their paper, they developed a dynamic programming algorithm which runs in pseudo-polynomial time for the general case and in polynomial time for the two special cases with equal processing times or equal tardiness penalties. Their model, however, does not include tardy deliveries or batch-setup time.

In this chapter, we study a single-customer problem which takes into account both batch-setup time and tardy deliveries. In Section 4.2, we first define the problem and then discuss some important propositions. In Section 4.3, for the general case, we propose a pseudo-polynomial algorithm, which shows that the problem is $\mathcal{NP}$-hard

only in the ordinary sense. The algorithm also runs in $O(n^5)$ time for two special cases with equal processing times or equal tardiness penalties. In Section 4.4, we convert the pseudo-polynomial algorithm into an FPTAS for the general case with running time $O(n^4/\varepsilon + n^4 \log n)$. Section 4.5 contains our concluding remarks.

4.2 Preliminaries

Note that all the terminologies, assumptions and notations introduced in Chapters 1 and 3 are applied in this chapter. Our goal is to find a schedule which minimizes the sum of the weighted number of tardy jobs and the batch-delivery costs, denoted by $1|s|\sum w_jU_j + bq$. Since the problem of minimizing weighted number of tardy jobs on a single machine, $1||\sum w_jU_j$, is $\mathcal{NP}$-hard [Karp, 1972], considering $s = 0$ and $q = 0$ in a $1|s|\sum w_jU_j + bq$ problem, we can see that the $1|s|\sum w_jU_j + bq$ problem is $\mathcal{NP}$-hard as well.

The following simple observations characterize the structure of optimal schedules we will search for.

Proposition 4.2.1 *There exists an optimal schedule for the* $1|s|\sum w_jU_j+bq$ *problem in which all early jobs are ordered in the earliest due date first order (EDD) within each batch.*

Proof. Since all early jobs in the same batch have the same completion times which are defined by the batch-completion time, the sequencing of jobs within a batch is immaterial. ■

Proposition 4.2.2 *There exists an optimal schedule for the* $1|s|\sum w_jU_j+bq$ *problem in which all tardy jobs (if any) are scheduled in the last batch (either in a tardy batch or in a mixed batch).*

Proof. Suppose that there is a tardy job in a batch which is scheduled before the last batch in an optimal schedule. If we move this job into this last batch, it will not increase the cost of the schedule. ■

Proposition 4.2.3 *There exists an optimal schedule for the* $1|s|\sum w_jU_j+bq$ *problem in which all early batches are scheduled in EDD order with respect to their batch-due date.*

Proof. Suppose that there are two early batches i and k in an optimal schedule with batch-completion times $C(i) < C(k)$ and batch-due dates $d(i) > d(k)$. Since all jobs in both batches are early, we have $d(i) > d(k) \geq C(k) > C(i)$. Thus if we schedule batch i after batch k, then all jobs in batch i and k are still early. Therefore, it will not increase the cost of the schedule. ■

Proposition 4.2.4 *There exists an optimal schedule for the* $1|s|\sum w_jU_j+bq$ *problem in which all early batches are scheduled in EDD order.*

Proof. Suppose that there are two early jobs i and k in an optimal schedule with completion times $C_i < C_k$ and due dates $d_i > d_k$. Since both jobs are early, we have $d_i > d_k \geq C_k > C_i$. Thus if we schedule job i into the same batch i where job k is scheduled, both jobs i and k are still early. Therefore, it will not increase the cost of the schedule. ■

Proposition 4.2.5 *There exists an optimal schedule for the* $1|s|\sum w_jU_j+bq$ *problem such that if the last batch of the schedule is not a tardy batch, then all jobs whose due dates are greater than or equal to the batch-completion time are scheduled in this last batch as early jobs.*

Proof. Let the batch-completion time of the last batch be t. Since the last batch is not a tardy batch, there must be at least one early job in this last batch whose due date is greater than or equal to t. If there is another job whose due date is greater than or equal to t but it was scheduled in an earlier batch, then we can simply move this job into this last batch without increasing the cost of the schedule. ■

Proposition 4.2.5 implies that the jobs which are first scheduled as tardy jobs can always be scheduled in the last batch when completing a partial schedule that contains only early jobs. Algorithm CH4-A1 we will present in the next section

uses this fact by generating all possible schedules on early jobs only and designating and putting aside the tardy jobs, which get scheduled only at the end in the last batch. It is important to note that when a job is designated to be tardy in a partial schedule, then its tardiness penalty is added to the cost of the partial schedule. We may encounter two situations: (1) We can schedule the jobs not yet scheduled and the jobs previously scheduled tardy by adding them to the last early batch if the resulting batch-completion time does not exceed the batch-due date of the last batch in the partial schedule, i.e., the jobs scheduled early in the last batch of the partial schedule remain early; (2) Start a new batch and put the jobs not yet scheduled and the jobs previously scheduled tardy in this last batch of the full schedule.

Putting all previously designated tardy jobs into the last batch can create a problem however: Some of these tardy jobs may end up being early in this last batch and thus the tardiness penalties of the tardy jobs in the cost of the partial schedule would have to be reduced by the tardiness penalties of these jobs. To do the cost adjustment correctly, however, we would have to know the due date and the tardiness penalty of all these jobs. This means that we would have to know exactly which jobs were designated tardy for the partial schedule. This would lead to an exponential increase in the complexity of Algorithm CH4-A1. In the next section, we show that for certain partial schedules, this problem would never occur, i.e., no previously scheduled tardy job can become early when adding it to the last batch that completes the partial schedule under either of the two situations mentioned above.

4.3 Pseudo-polynomial Algorithm

We know that tardy jobs can be delivered in the last batch, but setting them up in a separate batch could add the potentially unnecessary delivery cost q for this batch when in certain schedules it may be possible to deliver tardy jobs together with early jobs and save their delivery costs. The following Algorithm CH4-A1 gets around this problem using the concept of designated tardy jobs, whose batch assignment will be

determined only at the end. Without loss of generality, assume that the jobs are in EDD order, i.e., $d_1 \leq d_2 \leq ... \leq d_n$ and let $P = \sum_{j=1}^{n} p_j$. If $d_1 \geq P + s$, then it is easy to see that scheduling all jobs in a single batch will result in no tardy jobs, and this will be an optimal schedule. Therefore, we exclude this trivial case by assuming for the remainder of this chapter that some jobs are due before $P + s$. The state space used to represent a partial schedule in Algorithm CH4-A1 is described by five entries $\{k, l, t, d, v\}$:

k: the partial schedule is on the job set $\{1, 2, ..., k\}$;

l: the number of batches in the partial schedule;

t: the current batch-completion time of the last batch in the partial schedule;

d: the current batch-due date of the last batch in the partial schedule;

v: the cost of the partial schedule.

Before we describe Algorithm CH4-A1 in detail, let us consider how we can reduce the state space. Consider any two states (k, l, t_1, d, v_1) and (k, l, t_2, d, v_2). Without loss of generality, let $t_1 \leq t_2$. If $v_1 \leq v_2$, we can eliminate the second state because any later states which could be generated from the second state can not lead to better v value than the value of similar states generated from the first state. This validates the following elimination rule, and a similar argument could be used to justify the second remark.

Remark 4.3.1 *For any two states with the same entries $(k, l, t, d, \cdot)$, we can eliminate the state with larger v.*

Remark 4.3.2 *For any two states with the same entries $(k, l, \cdot, d, v)$, we can eliminate the state with larger t.*

Algorithm CH4-A1 recursively generates the states for the partial schedules on batches of early jobs and at the same time designates some other jobs to be

tardy without actually scheduling these tardy jobs. The jobs designated tardy will be added in the last batch at the time when the partial schedule gets completed into a full schedule. The tardiness penalty for every job designated tardy gets added to the state variable v at the time of designation. We look for an optimal schedule that satisfies the properties described in the propositions of the previous section. By Proposition 4.2.5, the tardy jobs should all be in the last batch of a full schedule. It is equivalent to say that any partial schedule $\{k, l, t, d, v\}$ with $1 \leq l \leq n-1$ can be completed into a full schedule by one of the following two ways:

- Simple Completion: Add all unscheduled jobs $\{k+1, k+2, ..., n\}$ and the previously designated tardy jobs to the last batch l, *if* the resulting batch-completion time $(P + ls)$ does not exceed the batch-due date d;
- Direct Completion: Open a new batch $l+1$, and add all unscheduled jobs $\{k+1, k+2, ..., n\}$ and the previously designated tardy jobs to the schedule in this batch.

We have to be careful, however, as putting a previously designated tardy job into the last batch this way may make such a job actually early if its completion time ($P + ls$ or $P + (l+1)s$, respectively) is not greater than its due date. This situation would require rescheduling such a designated tardy job among the early jobs and removing its tardiness penalty from the cost v. Unfortunately, such rescheduling is not possible, since we do not know the identity of the designated tardy jobs from the state variables (we could only derive their total length and tardiness penalty). The main insight behind our approach is that there are certain special states, that we will characterize, whose completion never requires such a rescheduling. We proceed with the definition of these special states.

It is clear that a full schedule containing exactly l $(1 \leq l \leq n)$ batches will have its last batch completed at $P + ls$. We consider all these possible completion times and define certain *marker jobs* m_i and corresponding *batch counters* g_i as follows: Let m_0 be the last job with $d_{m_0} < P + s$ and $m_0 + 1$ the first job with $d_{m_0+1} \geq P + s$.

If $m_0 + 1$ does not exist, i.e., $m_0 = n$, then we do not need to define any other marker jobs, all due dates are less than $P+s$, and we will discuss this case separately later. Otherwise, define $g_0 = 0$ and let $g_1 \geq 1$ be the largest integer for which $d_{m_0+1} \geq P + g_1 s$. Let the marker job associated with g_1 be job $m_1 \geq m_0 + 1$ whose due date is the largest due date strictly less than $P+(g_1+1)s$, i.e., $d_{m_1} < P+(g_1+1)s$ and $d_{m_1+1} \geq P + (g_1 + 1)s$. Define recursively for $i = 2, 3, ..., h-1$, $g_i \geq g_{i-1} + 1$ to be the smallest counter for which there is a marker job $m_i \geq m_{i-1} + 1$ such that $d_{m_i} < P + (g_i + 1)s$ and $d_{m_i+1} \geq P + (g_i + 1)s$. The last marker job is $m_h = n$ and its counter g_h is the largest integer for which $P + g_h s \leq d_{m_h} = d_n < P + (g_h + 1)s$. We also define $g_{h+1} = g_h + 1$. Since the maximum completion time to be considered is $P + ns$ for all possible schedules, any due dates which are greater than or equal to $P + ns$ can be reduced to $P + ns$ without affecting the solution. Thus we assume that $d_n \leq P + ns$, which also implies $g_h \leq n$.

For convenience, let us also define $T_{1,0} = P + g_1 s$, $T_{i,l} = P + (g_i + l)s$ for $i = 1, ..., h$ and $l = 0, 1, ..., l(i)$, where each $l(i)$ is the number for which $T_{i,l(i)} = P+(g_i+l(i))s = P+g_{i+1}s = T_{i+1,0}$. In particular, we have $l(h) \leq 1$ and $T_{h,l(h)} \leq T_{h,1} = P+(g_h+1)s \leq P+(n+1)s$. Note that this partitions the time horizon $[P, P+(g_h+1)s]$ into consecutive intervals of length s. We demonstrate these definitions in Figure 4.1.

We can distinguish the following two cases for these intervals:

1. $T_{i,1} = T_{i+1,0}$, i.e., $l(i) = 1$: This means that the interval immediately following $I_i = [T_{i,0}, T_{i,1})$ contains at least one due date. This implies that $g_{i+1} = g_i + 1$;

2. $T_{i,1} \neq T_{i+1,0}$, i.e., $l(i) > 1$: This means that there are $l(i) - 1$ intervals of length s starting at $P + (g_i + 1)s$ in which no due date is located.

In either case, it follows that every job $j > m_0$ has its due date in one of the intervals $I_i = [T_{i,0}, T_{i,1})$ for some $i \in \{1, ..., h\}$, and the intervals $[T_{i,l}, T_{i,l+1})$ contain no due date for $i = 1, ..., h-1$ and $0 < l < l(i)$.

Figure 4.1 shows that jobs from m_0+1 to m_1 have their due date in the interval $[T_{1,0}, T_{1,1})$. Each marker job m_i is the last job that has its due date in the interval $I_i = [T_{i,0}, T_{i,1})$ for $i = 1, ..., h$, i.e., we have $T_{i,0} \leq d_{m_{i-1}+1} \leq d_{m_{i-1}+2} \leq ... \leq d_{m_i} < T_{i,1}$.

Now let us group all jobs into $h+1$ non-overlapping job sets $G_0 = \{1, ..., m_0\}$, $G_1 = \{m_0 + 1, ..., m_1\}$ and $G_i = \{m_{i-1} + 1, ..., m_i\}$ for $i = 2, ..., h$. Then we have $d_j \in I_i$, $\forall j \in G_i$ and $i \geq 1$. We also define the job sets $J_0 = G_0$, $J_i = G_0 \cup G_1 \cup ... \cup G_i$, for $i = 1, 2, ..., h-1$ and $J_h = G_0 \cup G_1 \cup ... \cup G_h = J$.

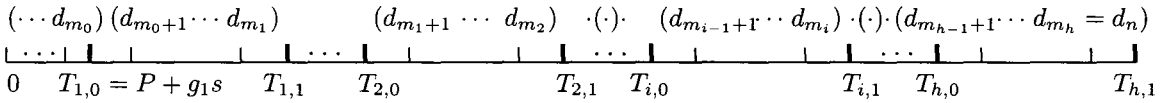


Figure 4.1: Marker jobs and corresponding intervals.

The special states for Algorithm CH4-A1 are defined by the fact that their (k, l) state variables belong to the set H defined as follows. If $m_0 = n$, then let $H = \{(n, 0), (n, 1), ..., (n, n-1)\}$. If $m_0 < n$, then let $H = H_1 \cup H_2 \cup H_3$, where $H_1 = \{(m_0, 1), (m_0, 2), ..., (m_0, g_1 - 1)\}$, $H_3 = \{(n, g_h), (n, g_h + 1), ..., (n, g_{h+1} - 1)\}$ and $H_2 = \{(m_1, g_1), (m_1, g_1 + 1), ..., (m_1, g_2 - 1), (m_2, g_2), ..., (m_i, g_i), (m_i, g_i + 1), ..., (m_i, g_{i+1} - 1), (m_{i+1}, g_{i+1}) ..., (m_{g_{h-1}}, g_{h-1}), ..., (m_{g_{h-1}}, g_h - 1)\}$. In particular, when $g_1 = 1$ we have $H_1 = \emptyset$ and when $g_h = 1$ or $g_h = n$ we have $H_3 = \emptyset$. Now we are ready to present Algorithm CH4-A1.

Algorithm CH4-A1

[Initialization] Start with jobs in EDD order and do the following:

1. Set $\mathcal{S}^0 = \{(0, 0, 0, 0, 0)\}$, $\mathcal{S}^{(k)} = \emptyset$, $k = 1, 2, ..., n+1$, $\mathcal{T}^* = \emptyset$, and determine m_0, g_i and m_i, $i = 1, 2, ..., h$;
2. If $m_0 = n$, then set $H = \{(n, 1), (n, 2), ..., (n, n-1)\}$; Otherwise set $H = H_1 \cup H_2 \cup H_3$.
3. Let $\Omega = \{(k, l) | 1 \leq l \leq k \leq n\}$ the set of all possible pairs and $\bar{H} = \Omega - H$, the complementary set of H.

[Generation] Generate set $\mathcal{S}^{(k)}$ from $\mathcal{S}^{(k-1)}$.

For $k = 1$ **to** $n + 1$

Set $\mathcal{T} = \emptyset$;

For each state $(k-1, l, t, d, v)$ in $\mathcal{S}^{(k-1)}$

Case $(k-1, l) \in H$

1. If $t < P + ls$, set $\mathcal{T}^* = \mathcal{T}^* \cup (n, l+1, P+(l+1)s, d', v+q)$ /*Generate the direct completion schedule by putting all remaining jobs and all designated tardy jobs into batch $l+1$, and add it to the solution set T^*, where d' is defined as the due date of the first job in batch $l+1$;
2. If $t = P + ls$, set $\mathcal{T}^* = \mathcal{T}^* \cup (n, l, P+ls, d, v)$ /*We have a partial schedule in which all jobs are early (only when $k-1=n$).

Case $(k-1, l) \in \bar{H}$

1. If $t + p_k \leq d$ and $k \leq n$, set $\mathcal{T} = \mathcal{T} \cup (k, l, t+p_k, d, v)$ /*Schedule job k as an early job in the current batch;
2. If $t + p_k + s \leq d_k$ and $k \leq n$, set $\mathcal{T} = \mathcal{T} \cup (k, l+1, t+p_k+s, d_k, v+q)$ /*Schedule job k as an early job in a new batch;
3. If $k \leq n$, set $\mathcal{T} = \mathcal{T} \cup (k, l, t, d, v+w_k)$ /*Designate job k as tardy by adding w_k to v and reconsider it at the end in direct completions.

Endfor

[Elimination] Update set $\mathcal{S}^{(k)}$.

1. For any two states (k, l, t, d, v) and (k, l, t, d, v') with $v \leq v'$, eliminate the one with v' from set $\mathcal{T}$ based on Remark 4.3.1;
2. For any two states (k, l, t, d, v) and (k, l, t', d, v) with $t \leq t'$, eliminate the one with t' from set $\mathcal{T}$ based on Remark 4.3.2;
3. Set $\mathcal{S}^{(k)} = \mathcal{T}$.

Endfor

[Result] Select the state with the smallest v in the set $\mathcal{T}^*$ as the optimal solution and trace back to obtain the optimal schedule.

We prove the correctness of Algorithm CH4-A1 by a series of lemmas, which establish the crucial properties for the special states.

Lemma 4.3.1 *Consider a partial schedule (m_i, l, t, d, v) on job set J_i, where $(m_i, l) \in H$. If its completion into a full schedule has $l+1$ batches, then the final cost of this completion is exactly $v+q$.*

Proof. We note that completing a partial schedule on l batches into a full schedule on $l+1$ batches means a direct completion, i.e., all the unscheduled jobs (the jobs in $J - J_i$, if any) and all the previously designated tardy jobs (if any) are put into batch $l+1$, with completion time $P + (l+1)s$.

Since all the previously designated tardy jobs are from J_i for a partial schedule (m_i, l, t, d, v), their due dates are not greater than $d_{m_i} < P + (g_i + 1)s \leq P + (l+1)s$. Therefore, all designated tardy jobs stay tardy when scheduled in batch $l+1$. Next we show that unscheduled jobs $j \in (J - J_i)$ must be early in batch $l+1$. We have three cases to consider.

Case 1. $m_0 = n$ and $i = 0$: In this case, $H = \{(n,1), (n,2), ..., (n, n-1)\}$ and $J_0 = J$, i.e., all jobs have been scheduled early or designated tardy in the state (m_0, l, t, d, v). Therefore, there are no unscheduled jobs.

Case 2. $m_0 < n$ and $l = g_i$: Since $g_0 = 0$ by definition, we must have $i \geq 1$ in this case. The first unscheduled job $j \in (J - J_i)$ is job $m_i + 1$ with due date $d_{m_i+1} \geq P + (g_i + 1)s = P + (l+1)s$. Thus job $m_i + 1$ and all other jobs from $J - J_i$ have a due date that is at least $P + (l+1)s$, and therefore they will all be early in batch $l+1$.

Case 3. $m_0 < n$ and $l > g_i$: This case is just an extension of the case of $l = g_i$. If $i = 0$, then the first unscheduled job for the state (m_0, l, t, d, v) is $m_0 + 1$. Thus every unscheduled job j has a due date $d_j \geq d_{m_0+1} \geq P + g_1 s \geq P + (l+1)s$, where the last inequality holds since $(m_0, l) \in H_1$ and therefore, $l \leq g_1 - 1$. If $1 \leq i < h$, then we cannot have $l(i) = 1$: By definition, if $l(i) = 1$, then $g_i + l(i) - 1 = g_i = g_{i+1} - 1$, which contradicts $l > g_i$ and $(m_i, l) \in H$. Therefore, we must have $l(i) > 1$, and l could be any value from $\{g_i + 1, ..., g_i + l(i) - 1\}$. This means that $P + (l+1)s \leq$

$P + (g_i + l(i))s = P + g_{i+1}s$. We know, however, that every unscheduled job has a due date that is at least $P + g_{i+1}s$ by the definition of H. Thus every job from $J - J_i$ will be early indeed. If $i = h$, then we have $m_h = n$ and $J_h = J$, and thus all jobs have been scheduled early or designated tardy in the partial schedule (m_i, l, t, d, v). Therefore, there are no unscheduled jobs.

In summary, we have proved that all previously designated tardy jobs (if any) remain tardy in batch $l + 1$, and all jobs from $J - J_i$ (if any) will be early. This means that v correctly accounts for the tardiness cost of the completed schedule, and we need to add to it only the delivery cost q for the additional batch $l + 1$. Thus the cost of the completed schedule is $v + q$ indeed. ■

Lemma 4.3.2 *Consider a partial schedule (m_i, l, t, d, v) on job set J_i, where $(m_i, l) \in H$ and $l \neq n-1$. Then any completion into a full schedule with more than $l+1$ batches has a cost that is at least $v+q$, i.e., the direct completion has the minimum cost among all such completions of (m_i, l, t, d, v).*

Proof. If $m_i = n$, then the partial schedule is of the form (n, l, t, d, v), $(n, l) \in H$ and $l \neq n - 1$. (This implies that either $m_0 = n$ with $i = 0$ or $(m_i, l) \in H_3$ with $i = h$.) Since there is no unscheduled job left, all the new batches in any completion are for previously designated tardy jobs. And since all the previously designated tardy jobs have due dates that are not greater than $d_n < P + (g_i + 1)s \leq P + (l + 1)s$, these jobs will stay tardy in the completion. The number of new batches makes no difference to the tardiness penalty cost of tardy jobs. Therefore, the best strategy is to open only one batch. Thus the final cost is minimum with cost $v + q$.

Consider now a partial schedule (m_i, l, t, d, v), $(m_i, l) \in H$ and $l \neq n - 1$ when $m_i < n$. Since all the previously designated tardy jobs (if any) are from J_i, their due dates are not greater than $d_{m_i} < P + (g_i + 1)s \leq P + (l + 1)s$. Furthermore, since all unscheduled jobs are from $J - J_i$, their due dates are not less than $d_{m_i+1} \geq P + g_{i+1}s \geq P + (l + 1)s$. Thus scheduling all of these jobs into batch $l + 1$ makes them early without increasing the tardiness cost. It is clear that this is the best we

can do for completing (m_i, l, t, d, v) into a schedule with $l+1$ or more batches. Thus the final cost of the direct completion is minimum again with cost $v+q$. ■

Lemma 4.3.3 *Consider a partial schedule (m_i, l, t, d, v) on job set J_i, where $i \geq 1$, $(m_i, l) \in H$ and $l > 1$. If it has a completion into a full schedule with exactly l batches and cost v', then (1) there must exist either a partial schedule $(m_i, l-1, \bar{t}, \bar{d}, \bar{v})$ whose direct completion is of the same cost v' or (2) (m_i, l, t, d, v) has a simple completion in which all unscheduled jobs can be scheduled early and all designated tardy jobs remain tardy when scheduled in batch l or (3) there exists a partial schedule $(m_{i-1}, l-1, \bar{t}, \bar{d}, \bar{v})$ whose direct completion is of the same cost v'.*

Proof. To complete the partial schedule (m_i, l, t, d, v) into a full schedule on l batches, all designated tardy jobs and unscheduled jobs have to be added into batch l.

Case 1. $l > g_i$: Let us denote the early jobs by $E_i \subseteq J_i$ in batch l in the partial schedule (m_i, l, t, d, v). Adding the designated tardy jobs and unscheduled jobs to batch l will result in a batch-completion time of $P+ls$. This makes all jobs in E_i tardy since $d_j \leq d_{m_i} < P + (g_i+1)s \leq P + ls$ for $j \in E_i$. Thus the cost of the full schedule should be $v' = v + \sum_{j \in E_i} w_j$. We cannot do this calculation, however, since there is no information available in Algorithm CH4-A1 about what E_i is. But if we consider the partial schedule $(m_i, l-1, \bar{t}, \bar{d}, \bar{v}) = (m_i, l-1, t - \sum_{j \in E_i} p_j, \bar{d}, v + \sum_{j \in E_i} w_j - q)$ with one less batch, where $\bar{d}$ is the smallest due date in batch $l-1$ in the partial schedule (m_i, l, t, d, v), the final cost of the direct completion of the partial schedule $(m_i, l-1, t - \sum_{j \in E_i} p_j, \bar{d}, v + \sum_{j \in E_i} w_j - q)$ would be exactly $v' = v + \sum_{j \in E_i} w_j$ by Lemma 4.3.1. We show next that this partial schedule $(m_i, l-1, t - \sum_{j \in E_i} p_j, \bar{d}, v + \sum_{j \in E_i} w_j - q)$ does get generated in the algorithm.

In order to see that Algorithm CH4-A1 will generate the partial schedule $(m_i, l-1, t - \sum_{j \in E_i} p_j, \bar{d}, v + \sum_{j \in E_i} w_j - q)$, suppose that during the generation of (m_i, l, t, d, v), Algorithm CH4-A1 starts batch l by adding job k as early. This implies that the jobs that Algorithm CH4-A1 designates as tardy on the path of states leading to (m_i, l, t, d, v) are in set $L_i = \{k, k+1, ..., m_i\} - E_i$. In other words, Algorithm CH4-A1 has a partial schedule $(k-1, l-1, t - \sum_{j \in E_i} p_j, \bar{d}, v - \sum_{j \in L_i} w_j - q)$

in the path of generation for (m_i, l, t, d, v). Then it will also generate from $(k-1, l-1, t-\sum_{j\in E_i} p_j, \bar{d}, v-\sum_{j\in L_i} w_j - q)$ the partial schedule $(m_i, l-1, t-\sum_{j\in E_i} p_j, \bar{d}, v+\sum_{j\in E_i} w_j - q)$ by simply designating all jobs in $E_i \cup L_i$ as tardy.

Case 2. $l = g_i \neq 1$: Suppose the partial schedule (m_i, l, t, d, v) has in batch l the sets of early jobs $E_{i-1} \cup E$, where $E_{i-1} \subseteq J_{i-1}$ and $E \subseteq (J_i - J_{i-1})$. Adding the designated tardy jobs and unscheduled jobs to batch l will result in a batch-completion time of $P+ls$. This makes all jobs in E_{i-1} tardy since $d_j \leq d_{m_{i-1}} < P+g_i s$ for $j \in E_{i-1}$. On the other hand, if $L \subseteq (J_i - J_{i-1} - E)$ denotes the previously designated tardy jobs from $J_i - J_{i-1}$ in (m_i, l, t, d, v), then these jobs become early since $P + g_i s \leq d_{m_{i-1}} + 1 \leq d_j$ for $j \in L$. For similar reasons, all previously designated tardy jobs not in L stay tardy, jobs in E remain early and all other jobs from $J - J_i$ will be early too. In summary, the cost for the full completed schedule derived from (m_i, l, t, d, v) should be $v' = v + \sum_{j\in E_{i-1}} w_j - \sum_{j\in L} w_j$. Again, we cannot do this calculation, since there is no information about E_{i-1} and L. However, suppose that $E_{i-1} \neq \emptyset$, and consider the partial schedule $(m_{i-1}, l-1, \bar{t}, \bar{d}, \bar{v}) = (m_{i-1}, l-1, t-\sum_{j\in E\cup E_{i-1}} p_j, \bar{d}, v+\sum_{j\in E_{i-1}} w_j - \sum_{j\in L} w_j - q)$ with one less batch, where $\bar{d}$ is the smallest due date in batch $l-1$ in the partial schedule (m_i, l, t, d, v). The final cost of the direct completion of the partial schedule $(m_{i-1}, l-1, t-\sum_{j\in E\cup E_{i-1}} p_j, \bar{d}, v+\sum_{j\in E_{i-1}} w_j - \sum_{j\in L} w_j - q)$ would be exactly $v' = v+\sum_{j\in E_{i-1}} w_j - \sum_{j\in L} w_j$ by Lemma 4.3.1. Next, we show that this partial schedule $(m_{i-1}, l-1, t-\sum_{j\in E\cup E_{i-1}} p_j, \bar{d}, v+\sum_{j\in E_{i-1}} w_j - \sum_{j\in L} w_j - q)$ does get generated during the execution of Algorithm CH4-A1.

To see the existence of the partial schedule $(m_{i-1}, l-1, \bar{t}, \bar{d}, \bar{v}) = (m_{i-1}, l-1, t-\sum_{j\in E\cup E_{i-1}} p_j, \bar{d}, v+\sum_{j\in E_{i-1}} w_j - \sum_{j\in L} w_j - q)$, note that Algorithm CH4-A1 must start batch l on the path of states leading to (m_i, l, t, d, v) by scheduling a job $k \leq m_{i-1}$ early in iteration k from a state $(k-1, l-1, t-\sum_{j\in E_i\cup E} p_j, \bar{d}, v-(\sum_{j=k}^{m_{i-1}} w_j - \sum_{j\in E_{i-1}} w_j) - \sum_{j\in L} w_j - q)$. (We cannot have $k > m_{i-1}$ since this would contradict $E_{i-1} \neq \emptyset$. Note also that $(\sum_{j=k}^{m_{i-1}} w_j - \sum_{j\in E_{i-1}} w_j)$ accounts for the tardiness penalty of those jobs from $\{k, k+1, ..., m_{i-1}\}$ that got designated tardy between iterations k and m_{i-1} during the generation of the state (m_i, l, t, d, v).) In this case, it is clear

that Algorithm CH4-A1 will also generate from $(k-1, l-1, t-\sum_{j\in E_i\cup E} p_j, \bar{d}, v-(\sum_{j=k}^{m_{i-1}} w_j - \sum_{j\in E_{i-1}} w_j) - \sum_{j\in L} w_j - q)$ a partial schedule on J_{i-1} in which all jobs in E_{i-1} are designated tardy, in addition to those jobs (if any) from $\{k, k+1, ..., m_{i-1}\}$ that are designated tardy in (m_i, l, t, d, v). Since this schedule will designate all of $\{k, k+1, ..., m_{i-1}\}$ tardy, the tardiness cost of this set of jobs must be added, which results in a state $(m_{i-1}, l-1, t-\sum_{j\in E\cup E_{i-1}} p_j, \bar{d}, v+\sum_{j\in E_{i-1}} w_j - \sum_{j\in L} w_j - q)$. This is the state $(m_{i-1}, l-1, \bar{t}, \bar{d}, \bar{v})$ whose existence we claimed.

The remaining case is when $E_{i-1} = \emptyset$. In this case, batch l has no early jobs in the partial schedule (m_i, l, t, d, v) from the set J_{i-1}, $k = m_i$ is the only member of the set E, $L = \emptyset$ and $v = v'$. Therefore, there are no early or tardy jobs that would need to be rescheduled. Furthermore, adding all unscheduled jobs, which are in $J - J_i$, will make them early. Thus (m_i, l, t, d, v) can be simply completed into a full schedule with cost v. ■

The remaining special cases of $l = 1$, which are not covered by the preceding lemmas are $(m_i, l) = (m_1, 1)$ or $(m_i, l) = (m_0, 1)$: Since all jobs are delivered at the same time $P + s$, all jobs in J_0 or J, respectively, are tardy, and the rest of the jobs are early. Thus there is only one possible full schedule with cost $v' = \sum_{j=1}^{m_0} w_j + q$ or $v' = \sum_{j=1}^{n} w_j + q$. In summary, consider any partial schedule (m_i, l, t, d, v) on job set J_i, where $(m_i, l) \in H$, or a partial schedule (n, l, t, d, v) on job set J and assume that the full schedule $(n, l', P + l's, d', v')$ is a completion of this partial schedule and has minimum cost v'. Then the following schedules generated by Algorithm CH4-A1 will contain a schedule with the same minimum cost as $(n, l', P + l's, d', v')$:

1. the direct completion of a partial schedule (m_i, l, t, d, v), if $(m_i, l) \neq (m_i, g_i)$ and $l' > l$, by Lemma 4.3.1 and Lemma 4.3.2;

2. the direct completion of a partial schedule $(m_i, l-1, \bar{t}, \bar{d}, \bar{v})$, if $(m_i, l) \neq (m_i, g_i)$, and $l' = l$, by Lemma 4.3.3;

3. the direct completion of a partial schedule $(m_{i-1}, l-1, \bar{t}, \bar{d}, \bar{v})$, if $(m_i, l) = (m_i, g_i)$, $i > 1$ and $l' = l$, by Lemma 4.3.3;

4. the full schedule $(n, 1, P+s, d_{m_0+1}, \sum_{j=1}^{m_0} w_j + q)$ if $m_0 < n$ and $l' \geq l = g_1 = 1$ i.e., $(m_i, l) = (m_1, 1)$;

5. the full schedule $(n, 1, P+s, d_1, \sum_{j=1}^{n} w_j + q)$, if $m_0 = n$ and $l' \geq l = 1$. i.e., $(m_i, l) = (m_0, 1)$.

Theorem 4.3.1 *Algorithm CH4-A1 finds an optimal solution for the* $1|s|\sum w_jU_j+bq$ *problem in* $O(n^3 \min\{d_n, P+ns, W+nq\})$ *time, where* $P = \sum_{j=1}^{n} p_j$ *and* $W = \sum_{j=1}^{n} w_j$. *This shows that the problem is* $\mathcal{NP}$*-hard only in the ordinary sense.*

Proof. The correctness of Algorithm CH4-A1 follows directly from the previous discussion and Lemmas 4.3.1, 4.3.2 and 4.3.3.

The time complexity of Algorithm CH4-A1 is dominated by the [Generation] procedure. At the beginning of iteration k, the total number of possible values for the state variables $\{k, l, t, d, v\}$ in $\mathcal{S}^{(k)}$ is upper bounded as follows: n is the upper bound of k and l; n is the upper bound for the number of different d values; $\min\{d_n, P+ns\}$ is an upper bound of t and $W+nq$ is an upper bound of v, and because of the elimination rules, $\min\{d_n, P+ns, W+nq\}$ is an upper bound for the number of different combinations of t and v. Thus the total number of different states at the beginning of each iteration in the [Generation] procedure is at most $O(n^2 \min\{d_n, P+ns, W+nq\})$. In each iteration k, there are at most three new states generated from each state in $\mathcal{S}^{(k-1)}$ and this takes constant time. Since there are n iterations, the [Generation] procedure could indeed be done in $O(n^3 \min\{d_n, P+ns, W+nq\})$ time. ■

Corollary 4.3.1 *For the* $1|s|\sum w_jU_j + bq$ *problem, if all jobs have equal tardiness penalties, i.e.,* $w_j = w > 0$, $\forall j \in J$, *then Algorithm CH4-A1 finds an optimum solution in* $O(n^5)$ *time.*

Proof. For any state, v is the sum of two different cost components: the delivery costs from $\{q, 2q, ..., nq\}$ and the weighted number of tardy jobs from $\{0, w, ..., nw\}$. Therefore, v can take at most $n(n+1)$ different values and the upper bound for the number of different states becomes $O(n^3 \min\{d_n, P+ns, n^2\}) = O(n^5)$. ■

Corollary 4.3.2 *For the* $1|s|\sum w_jU_j + bq$ *problem, if all jobs have equal processing times, i.e.,* $p_j = p > 0$, $\forall j \in J$, *then Algorithm CH4-A1 finds an optimum solution in* $O(n^5)$ *time.*

Proof. For any state, t is the sum of two different time components: the setup times from $\{s, ..., ns\}$ and the processing times from $\{0, p, ..., np\}$. Thus, t can take at most $n(n+1)$ different values, and the upper bound for the number of different states becomes $O(n^3 \min\{d_n, n^2, W + nq\}) = O(n^5)$. ■

4.4 Fully Polynomial Time Approximation Scheme

To develop an FPTAS, we will use static interval partitioning originally suggested by Sahni [1976] for maximization problems. The efficient implementation of this approach for minimization problems is more difficult, as it requires prior knowledge of a lower and upper bound for the unknown optimum value v^*, such that the upper bound is a constant multiple of the lower bound.

4.4.1 Initial Bounds

In order to obtain such a pair of bounds, using the same data, we construct an auxiliary problem in which we want to minimize the maximum weight of tardy jobs and batches have the same batch-setup time s but with zero batch-delivery cost. We denote this problem by $1|s|\max w_jU_j$. We will prove that the optimal solution value for the $1|s|\max w_jU_j$ problem will be a lower bound for v^*, the optimal solution value for the $1|s|\sum w_jU_j + bq$ problem. This enables us to determine the initial bounds.

To solve the $1|s|\max w_jU_j$ problem, we first sort all jobs into smallest-weight-first order, i.e., $w_{[1]} \leq w_{[2]} \leq ... \leq w_{[n]}$. Here we are using $[k]$ to denote the job with the k-th smallest weight. Suppose that $[k^*]$ has the largest weight among the tardy jobs in an optimal schedule. Since we can always reschedule these tardy jobs at the end of the optimal schedule without making its cost worse, there is also an optimal

schedule in which every job $[i]$, for $i = 1, 2, ..., k^*$, is at the end of the schedule. We also can assume without loss of generality that the early jobs are scheduled in EDD order in an optimal schedule. Thus we can restrict our search for an optimal schedule of the following form: There is a $k \in \{0, 1, ..., n\}$ such that jobs $\{[k+1], ..., [n]\}$ are early and they are scheduled in EDD order in the first part of the schedule, followed by jobs $\{[1], [2], ..., [k]\}$ in the last batch. The existence of such a schedule can be verified by the following Algorithm CH4-A2(k).

Algorithm CH4-A2(k)

[Initialization] For a given k value, sort the jobs $\{[k+1], ..., [n]\}$ into EDD order, and let this sequence be $(\theta_1, \theta_2, ..., \theta_f)$, where $f = n - k$.

1. Set $i = 1$, $j = \theta_1$, $t = s + p_j$ and $d = d_j$.
2. If $t > d$, then no feasible schedule exists and go to [Report];
3. If $t \leq d$, then set $i = 2$ and go to [FeasibilityChecking].

[FeasibilityChecking] Try to schedule job i into a partial schedule.

While $i \leq f$, **set** $j = \theta_i$

1. If $t + p_j > d$, then start a new batch for job j
   (a) If $t + s + p_j > d_j$, then no feasible schedule exists and go to [Report];
   (b) If $t + s + p_j \leq d_j$, then set $t = t + s + p_j$, $d = d_j$, $i = i + 1$ and go to [FeasibilityChecking].
2. If $t + p_j \leq d$, then set $t = t + p_j$, $i = i + 1$ and go to [FeasibilityChecking].

Endwhile

[Report] If $i \leq f$, then no feasible schedule exists; Otherwise, there exists a feasible schedule in which all jobs $\{\theta_1, \theta_2, ..., \theta_f\}$ are early.

Theorem 4.4.1 *Algorithm CH4-A2(k) reports the existence of no feasible schedule or of a feasible schedule for the* $1|s|\max w_jU_j$ *problem on job set* $\{[k+1],...,[n]\}$ *in* $O(n\log n)$ *time.*

Proof. Algorithm CH4-A2(k) goes through the [FeasibilityChecking] procedure at most n times and each time it needs at most four operations. Therefore, for a given EDD sequence, Algorithm CH4-A2(k) reports in $O(n)$ time. However, for $k = 0$, Algorithm CH4-A2(k) constructs the EDD sequence on the whole job set J, which requires $O(n\log n)$ time. Thus the overall running time is $O(n\log n)$. ■

The $1|s|\max w_jU_j$ problem can be solved by repeatedly calling Algorithm CH4-A2(k) in a standard binary search to find the smallest k value, denoted by k^*, for which Algorithm CH4-A2(k) returns that a feasible schedule exists. Assume that all jobs are indexed as: $w_{[1]} \leq w_{[2]} \leq ... \leq w_{[n]}$.

Algorithm CH4-A3

[Initialization] Set $k_1 = 0$, $k_2 = n$ and $k^* = 0$.

While $k_1 < k_2$, **do**

1. Set $k = \lceil (k_1 + k_2)/2 \rceil$ and run Algorithm CH4-A2(k);
2. If it reports that no feasible schedule exists, then set $k_1 = k$;
3. If it reports a feasible schedule, then set $k^* = k$ and $k_2 = k$.

Endwhile

[Report] If $k^* = 0$, then $w^* = 0$; Otherwise, $w^* = w_{[k^*]}$.

Theorem 4.4.2 *Algorithm CH4-A3 finds an optimal solution to the* $1|s|\max w_j\ U_j$ *problem in* $O(n\log^2 n)$ *time.*

Proof. Since Algorithm CH4-A3 does a binary search, Algorithm CH4-A2(k) can be called at most $O(\log n)$ times. By Theorem 4.4.1, the overall running time is $O(n\log^2 n)$. ■

Corollary 4.4.1 *The optimal solution value v^* for the $1|s|\sum w_jU_j + bq$ problem is in the interval $[v', nv']$, where $v' = w^* + q$ and w^* is the optimal solution value for the $1|s|\max w_jU_j$ problem.*

Proof. Since there is at least one batch in any feasible schedule for the $1|s|\sum w_jU_j + bq$ problem, $v' = w^* + q$ is a lower bound for v^*. If there is a feasible schedule for the $1|s|\max w_jU_j$ problem on job set $\{[k^*+1], ..., [n]\}$, then there is a feasible schedule for the $1|s|\sum w_jU_j + bq$ problem with at most $n - k^* + 1$ batches on job set $\{1, ..., n\}$. Since the tardiness penalty of tardy jobs in such a schedule is at most k^*w^*, we have the upper bound $k^*w^* + (n - k^* + 1)q$. Therefore $nv' = nw^* + nq \geq k^*w^* + (n - k^* + 1)q$ is an upper bound of v^* as well. ■

4.4.2 Tight Bounds

In order to narrow the initial bounds $[v', nv']$, we first propose Algorithm CH4-A4(u, ε), which for given u and ε, either returns a schedule with cost $v \leq u$ or verifies that $(1 - \varepsilon)u$ is a lower bound for the cost of any solution. The algorithm is very similar to Algorithm CH4-A1 with a certain variation of the [Elimination] and [Result] procedures.

Algorithm CH4-A4(u, ε)

[Initialization] Do the same as in Algorithm CH4-A1.

[Partitioning] Partition the interval $[0, u]$ into $\lceil n/\varepsilon \rceil$ equal intervals of size $u\varepsilon/n$, with the last one possibly smaller.

[Generation] Generate set $\mathcal{S}^{(k)}$ from $\mathcal{S}^{(k-1)}$.

For $k = 1$ **to** $k = n + 1$

Set $\mathcal{T} = \emptyset$;

For each state $(k-1, l, t, d, v)$ in $\mathcal{S}^{(k-1)}$

Do the same as in Algorithm CH4-A1.

Endfor

[Elimination] Update set $\mathcal{S}^{(k)}$.

1. Eliminate any state (k, l, t, d, v) if $v > u$.
2. If more than one state has a v value that falls into the same interval, then discard all but one of these states, keeping only the representative state with the smallest t coordinate for each interval.
3. For any two states (k, l, t, d, v) and (k, l, t, d, v') with $v \leq v'$, eliminate the one with v' from set $\mathcal{T}$ based on Remark 4.3.2;
4. Set $\mathcal{S}^{(k)} = \mathcal{T}$.

Endfor

[Result] If $\mathcal{T}^* = \emptyset$, then $v^* > (1 - \varepsilon)u$; Otherwise $v^* \leq u$.

Theorem 4.4.3 *If Algorithm CH4-A4(u, ε) returns with $\mathcal{T}^* = \emptyset$, then $v^* > (1-\varepsilon)u$; otherwise $v^* \leq u$. The complexity of Algorithm CH4-A4(u, ε) is $O(n^4/\varepsilon)$.*

Proof. If $\mathcal{T}^*$ is not empty, then there is at least one state (n, l, t, d, v) with v in some subinterval of $[0, u]$ that has not been eliminated. Therefore, we have $v^* \leq v \leq u$.

If $\mathcal{T}^* = \emptyset$, then all states with the first two entries $(k, l) \in H$ have been eliminated. Consider any feasible schedule (n, l, t, d, v). The fact that $\mathcal{T}^* = \emptyset$ means that any ancestor state of (n, l, t, d, v) with cost $\tilde{v} \leq v$ must have been eliminated at some iteration k in the algorithm either because $\tilde{v} > u$ or by interval partitioning, which kept some other representative state with cost $\tilde{v}'$ and maximum error $\varepsilon u/n$. In the first case, we also have $v > u$. In the second case, let $v' \geq \tilde{v}'$ be the cost of a completion of the representative state and we must have $v' > u$ since $\mathcal{T}^* = \emptyset$. Since the error introduced in one iteration is at most $\varepsilon u/n$, the overall error is at most $n(\varepsilon u/n) = \varepsilon u$, i.e., $v \geq v' - n(\varepsilon u/n) = v' - \varepsilon u > u - \varepsilon u = (1-\varepsilon)u$. Thus $v > (1-\varepsilon)u$ for any feasible cost value v.

Notice that $|\mathcal{S}^{(k)}| \leq \lceil n^3/\varepsilon \rceil$ for $k = 1, 2, ..., n$. Since all operations on a single state can be performed in $O(1)$ time, the overall complexity is $O(n^4/\varepsilon)$. ■

Next, we consider to use $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ introduced in Chapter 3 to narrow the bounds $[v', nv']$. Let $a_1 = v'$, $a_2 = nv'$, $\omega = u$ and $\mathcal{P}$ be the $1|s|\sum w_j U_j + bq$ problem. Then Algorithm CH4-A4$(u, 1/3)$ can be for $\mathcal{A}(\omega, \mathcal{P})$ in $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$. Since Algorithm CH4-A4$(u, 1/3)$ runs in $O(n^4)$ time, then $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ reports ξ that implies a pair of tight bounds $[\xi, 3\xi]$ in $O(n^4 \log\log n)$ time. In this situation, we refer to $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ as Algorithm CH4-A5.

Corollary 4.4.2 *Algorithm CH4-A5 can narrow the bounds* $[v', nv']$ *into* $[\bar{v}, 3\bar{v}]$ *in* $O(n^4 \log\log n)$ *time by setting* $\bar{v} = \xi$.

4.4.3 Approximation

For any given $\varepsilon > 0$, using the bounds $\bar{v} \leq v^* \leq 3\bar{v}$ obtained by Algorithm CH4-A5, we run a slightly changed version of Algorithm CH4-A4(u, ε), called Algorithm CH4-A6, with $u = (1 + \varepsilon/3)3\bar{v}$: The only difference is that in the [Partitioning] step we partition $[0, u] = [0, (1+\varepsilon/3)3\bar{v}]$ into $n\lceil 3/\varepsilon + 1 \rceil$ intervals of size at most $\varepsilon\bar{v}/n$, so that the cumulative error over n iterations will be no more than $\varepsilon\bar{v}$. Since we know that the problem has an optimal solution value $v^* \leq 3\bar{v}$, the algorithm will find a solution v for the $1|s|\sum w_j U_j + bq$ problem such that $v \leq 3\bar{v} + \varepsilon\bar{v} = u$, which means that the algorithm will never end with an empty $\mathcal{T}^*$. Furthermore, whichever subinterval of $[\bar{v}, 3\bar{v}]$ v^* falls into, the algorithm will generate an approximate solution v with at most $\varepsilon\bar{v}$ error away from it, i.e., $v \leq v^* + \varepsilon\bar{v} \leq (1 + \varepsilon)v^*$. Therefore, we have the following corollary.

Corollary 4.4.3 *For any given* $\varepsilon > 0$, *Algorithm CH4-A6 finds a* $(1+\varepsilon)$*-approximate solution for the* $1|s|\sum w_j U_j + bq$ *problem in* $O(n^4/\varepsilon)$ *time.*

Now we are ready to present the final Algorithm CH4-A7, which combines into an FPTAS the previous algorithms as subroutines.

Algorithm CH4-A7

[InitialBounds]: Run Algorithm CH4-A3 and set $v' = q + w^*$;

[TightBounds]: Run Algorithm CH4-A5 and obtain $\bar{v}$ such that $\bar{v} \leq v^* \leq 3\bar{v}$.

[Approximation]: Run Algorithm CH4-A6 and obtain an approximate schedule.

Theorem 4.4.4 *For the* $1|s|\sum w_jU_j+bq$ *problem, Algorithm CH4-A7 finds a* $(1+\varepsilon)$*-approximate solution in* $O(n^4/\varepsilon + n^4 \log\log n)$ *time.*

Proof. The complexity and correctness follows from the preceding discussion and the complexity of the component algorithms. ■

4.5 Summary

In this chapter, we studied the $1|s|\sum w_jU_j + bq$ problem. We first proved that it is $\mathcal{NP}$-hard. Then we further classified that the problem is $\mathcal{NP}$-hard in the ordinary sense by presenting a pseudo-polynomial algorithm, which runs in polynomial time for the case with equal processing times and the case with equal tardiness penalties. Finally, we converted the pseudo-polynomial algorithm into an FPTAS for the general problem.

Chapter 5

Multiple-customer Model with Delivery Costs

In this chapter, we study a supply chain scheduling problem with delivery costs for m customers, where m is a fixed number and independent on the size of the model. Notice that the single-customer model in Chapter 4 is a special case with $m = 1$.

5.1 Introduction

In practice, a supplier usually maintains a relatively stable set of customers. In this situation, all the jobs produced by the supplier are divided into groups with respect to the customers. Therefore, we extend the single-customer problem studied in Chapter 4 into a multiple-customer problem which will be defined in detail in Section 5.2. Hall and Potts [2003] have studied a similar model, but without batch-setup time or tardy deliveries.

This chapter is organized as follows. In Section 5.2, we first define the problem and then discuss some important propositions. In Section 5.3, we propose a pseudo-polynomial algorithm which proves that the problem is $\mathcal{NP}$-hard only in the ordinary

sense. In Section 5.4, we study a restricted version of the problem, where all tardy jobs (if any) have to be delivered in separate batches. With this restriction, we are able to find a pseudo-polynomial algorithm and an FPTAS for it. In Section 5.5, we prove that the solution produced by the FPTAS for the restricted problem is near-optimal for our original problem. Section 5.6 contains our concluding remarks.

5.2 Preliminaries

Note that all the terminologies, assumptions and notations introduced in Chapters 1 and 3 are applied in this chapter. Our goal is to find a schedule which minimizes the sum of the weighted number of tardy jobs and the batch-delivery costs, denoted by $1|s_i|\sum w_{(i,k)}U_{(i,k)} + b_iq_i$. Since the $1|s|\sum w_jU_j + bq$ problem has already been shown to be $\mathcal{NP}$-hard, we know that the $1|s_i|\sum w_{(i,k)}U_{(i,k)} + \sum b_iq_i$ problem is $\mathcal{NP}$-hard as well. The following easy-to-prove propositions can be seen as extensions of Proposition 4.2.1, 4.2.2, 4.2.3 and 4.2.5 developed for the $1|s|\sum w_jU_j + bq$ problem in Chapter 4.

Proposition 5.2.1 *There exists an optimal schedule for the* $1|s_i|\sum w_{(i,k)}U_{(i,k)}+\sum b_iq_i$ *problem such that jobs in the same batch are processed without any interruption by any job from another batch.*

Proof. There is no benefit from interrupting the processing of jobs in a batch by jobs from another batch. ■

Proposition 5.2.2 *There exists an optimal schedule for the* $1|s_i|\sum w_{(i,k)}U_{(i,k)}+\sum b_iq_i$ *problem in which early jobs are ordered in EDD order within any batch.*

Proof. Notice that the earliness of a job is determined by the batch-completion time and its due date. Thus any order, and therefore the EDD order produces the same set of early jobs within a batch. ■

Proposition 5.2.3 *There exists an optimal schedule for the* $1|s_i|\sum w_{(i,k)}U_{(i,k)}+\sum b_iq_i$ *problem in which the early batches are ordered in EDD order with respect to the batch-due date.*

Proof. Suppose that there are two early batches: batch i and batch k with batch-completion times $C(i) < C(k)$, and batch-due dates $d(i) > d(k)$. Since the early jobs are on time, we have $d(i) > d(k) \geq C(k) > C(i)$. Thus if we simply move batch i after batch k in the schedule, the early jobs in the original schedule remain early. ■

Proposition 5.2.4 *There exists an optimal schedule for the* $1|s_i|\sum w_{(i,k)}U_{(i,k)}+\sum b_iq_i$ *problem in which the early jobs for the same customer are ordered in EDD order over all early batches for this customer.*

Proof. Suppose that there are two early batches for the same customer, batch i and batch k with batch-completion times $C(i) < C(k)$. If in batch i there is an early job whose due date is greater than the due date of another early job in batch k, then this job can be moved from batch i to batch k without increasing the cost of the schedule. After we moved all such jobs into the later batches, we can apply Proposition 5.2.2 to obtain the desired schedule. ■

Proposition 5.2.5 *There exists an optimal schedule for the* $1|s_i|\sum w_{(i,k)}U_{(i,k)}+\sum b_iq_i$ *problem in which the tardy jobs for customer* i, $i = 1, ..., m$ *are delivered either in the last early batch (which may become a mixed batch) for customer* i *or in the designated tardy batch for the customer.*

Proof. If the optimal schedule has some tardy jobs in the designated tardy batch for the customer, then we can move all other tardy jobs for this customer into the designated batch without increasing the cost of the schedule. If the optimal schedule has some tardy jobs in the last (mixed) batch, then we can move all other tardy jobs from any earlier scheduled batch for this customer into the last batch without increasing the cost of the schedule. ■

Proposition 5.2.5 implies that only the last early batch can become a mixed batch for a customer. Thus, when scheduling jobs for a customer i, we can restrict our search to two types of schedules:

- Type I All tardy jobs are scheduled in the designated tardy batch for customer i (and all previous batches for the same customer are early);
- Type II All tardy jobs are scheduled at the end of the last early batch (which becomes a mixed batch) for customer i, thereby saving the delivery cost for the designated tardy batch, which is empty. This also implies that job (i, n_i) is always scheduled in the last batch for customer i in a Type II schedule.

5.3 Pseudo-polynomial Algorithm

The state space used to represent a partial schedule in the following Algorithm CH5-A1 is described by six entries $\{\mathcal{J}^k, \mathcal{H}, t, h, d, v\}$ defined as follows:

$\mathcal{J}^k = (k_1, \cdots, k_m)$ records that the first k_i, $i \in M$ jobs have been scheduled in this partial schedule, where $k = \sum_{i \in M} k_i$ and $M = \{1, ..., m\}$;

$\mathcal{H} = (\tau_1, \tau_2, \cdots, \tau_m)$: the partial schedule where τ_i is the total length of jobs scheduled in the designated tardy batch for customer i;

t: the current batch-completion time of the current batch in this partial schedule;

h: the current batch of this partial schedule is for customer h, $h \in M$;

d: the current batch-due date of the current batch in this partial schedule;

v: the recursively calculated objective value for this partial schedule.

Assume that jobs in each set J_i are in EDD order. Algorithm CH5-A1 starts from the empty state $(\mathcal{J}^0, \mathcal{H}, 0, 0, 0, 0)$, where no job has been scheduled yet with $\mathcal{J}^0 = (0, ..., 0)$ and $\mathcal{H}$ =$(0, ..., 0)$. The set $\mathcal{S}^{(k)}$ contains all generated partial schedules

with k jobs, $k = 1, 2, \cdots, n$. First, Type I partial schedules are generated in non-decreasing order of their cardinality k. All these schedules are implicitly assumed to have a designated tardy batch for each customer i at the end of the schedule, that is not empty when $\tau_i > 0$. By Proposition 5.2.5, we know that when we want to obtain a Type II schedule for customer i, we only need to consider the last early batch for customer i, which must contain job (i, n_i). Thus we consider rescheduling all tardy jobs for customer i into an early batch only when we are scheduling the job (i, n_i).

Algorithm CH5-A1

[Initialization] Set $\mathcal{S}^{(0)} = \{(\mathcal{J}^0, \mathcal{H}, 0, 0, 0, 0)\}$, $\mathcal{H} = (0, ..., 0)$ and $\mathcal{S}^{(k)} = \emptyset$, $k = 1, ..., n$.

[Generation] Generate set $\mathcal{S}^{(k)}$ from $\mathcal{S}^{(k-1)}$.

For $k = 1$ **to** n

For each $(\mathcal{J}^{k-1}, \mathcal{H}, t, h, d, v) \in \mathcal{S}^{(k-1)}$ with $k - 1 = \sum_{k_i \in \mathcal{J}^{k-1}} k_i$

For each $i \in M$ with $k_i \neq n_i$, set $k_i' = k_i + 1$ and $\mathcal{J}^k = (k_1, \cdots, k_i', \cdots, k_m)$

1. If $t + p_{(i,k_i')} \leq d$ and $i = h$, then set $t' = t + p_{(i,k_i')}$ and $\mathcal{S}^{(k)} = \mathcal{S}^{(k)} \cup (\mathcal{J}^k, \mathcal{H}, t', h, d, v)$ */*Schedule job (i, k_i') as an early job in the current batch;*

2. If $t + p_{(i,k_i')} + s_i \leq d_{(i,k_i')}$, then set $t' = t + p_{(i,k_i')} + s_i$, $h' = i$, $d' = d_{(i,k_i')}$, $v' = v + q_i$ and $\mathcal{S}^{(k)} = \mathcal{S}^{(k)} \cup (\mathcal{J}^k, \mathcal{H}, t', h', d', v')$ */*Schedule job (i, k_i') as an early job in a new batch for customer i;*

3. If $\tau_i > 0$, then set $\tau_i' = \tau_i + p_{(i,k_i')}$ and $\tau_j' = \tau_j$ for $j \neq i$ to define $\mathcal{H}'$, $v' = v + w_{(i,k_i')}$ and $\mathcal{S}^{(k)} = \mathcal{S}^{(k)} \cup (\mathcal{J}^k, \mathcal{H}', t, h, d, v')$ */*Schedule job (i, k_i') as a tardy job in the nonempty designated tardy batch for customer i;*

4. If $\tau_i = 0$, then set $\tau_i' = p_{(i,k_i')}$ and $\tau_j' = \tau_j$ for $j \neq i$ to define $\mathcal{H}'$, $v' = v + w_{(i,k_i')} + q_i$ and $\mathcal{S}^{(k)} = \mathcal{S}^{(k)} \cup (\mathcal{J}^k, \mathcal{H}', t, h, d, v')$ */*Schedule*

job (i, k_i') *a tardy job by putting it into a new designated tardy batch for customer* i*;*

5. If $k_i' = n_i$, $i = h$, $\tau_i > 0$, and $t + p_{(i,k_i')} + \tau_i \leq d$, then set $t' = t + p_{(i,k_i')} + \tau_i$, $\tau_i' = 0$ and $\tau_j' = \tau_j$ for $j \neq i$ to define $\mathcal{H}'$, $v' = v - q_i$ and $\mathcal{S}^{(k)} = \mathcal{S}^{(k)} \cup (\mathcal{J}^k, \mathcal{H}', t', h, d, v')$ */*Reschedule job* (i, k_i') *and the tardy jobs from the designated tardy batch for customer* i *into the current batch;*
6. If $k_i' = n_i$, $\tau_i > 0$, and $t + p_{(i,k_i')} + \tau_i + s_i \leq d_{(i,k_i')}$, then set $t' = t + p_{(i,k_i')} + \tau_i + s_i$, $\tau_i' = 0$ and $\tau_j' = \tau_j$ for $j \neq i$ to define $\mathcal{H}'$, $d' = d_{(i,k_i')}$ and $\mathcal{S}^{(k)} = \mathcal{S}^{(k)} \cup (\mathcal{J}^k, \mathcal{H}', t', h, d', v)$ */*Reschedule job* (i, k_i') *and the tardy jobs from the designated tardy batch for customer* i *into a newly started mixed batch for customer* i*;*

Endfor

Endfor

Endfor

[Result] Select the state with the smallest v from set $\mathcal{S}^{(n)}$ and trace back to obtain an optimal schedule.

In steps 5 and 6, where we reschedule all tardy jobs for customer i together with job (n_i, i) into the last mixed batch for customer i, we update $v' = v - q_i$ and $v' = v$, respectively. This will account correctly for the delivery component for our cost, since we are reducing the number of batches for customer i by one in the first case, and it does not change in the second case (we empty the designated tardy batch, but we set up a new mixed batch). When we do this rescheduling, however, some of the rescheduled tardy jobs from the designated tardy batch may become early again. It means that the resulting v' value may be greater than the true objective value for this partial schedule, since the objective value would need to be reduced by the tardiness penalty of any previously tardy job that becomes early. To do this

adjustment correctly, we would also need to store for each partial schedule in its state the total tardiness penalty of tardy jobs in its designated tardy batch for each customer, which would add an additional factor of W^m to the size of the state space, where $W = \sum_{(i,k)\in J} w_{(i,k)}$. We show below, however, that this can be avoided.

Lemma 5.3.1 *Suppose that during the execution of rescheduling steps* 5 *or* 6 *in Algorithm CH5-A1, a state* $(\mathcal{J}^k, \mathcal{H}, t, h, d, v)$ *is generated for partial schedule* σ *in which some previously tardy jobs become early and thus* v *does not correctly represent the tardiness cost of the corresponding schedule* σ*. Then the algorithm must also generate a state* $(\mathcal{J}^k, \mathcal{H}, t, h, d, v'')$ *and schedule* σ'' *such that* v'' *correctly represents the cost of* σ''*, the schedules* σ *and* σ'' *have the same early and tardy job sets and* $v'' < v$*.*

Proof. Consider a partial schedule σ and state $(\mathcal{J}^k, \mathcal{H}, t, h, d, v)$ generated in steps 5 or 6, which is demonstrated in Figure 5.1 below. (The batch boundaries are indicated by longer and heavier lines.)

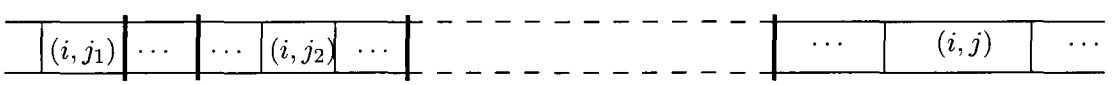


Figure 5.1: A partial schedule: (J^k, T, t, h, d, v).

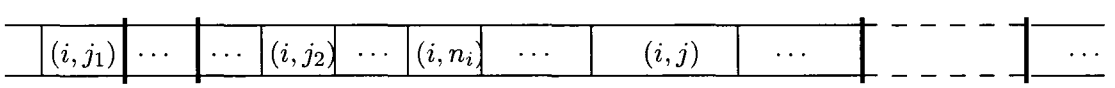


Figure 5.2: A full schedule σ with job (i, j) early.

Suppose job (i, j) is first scheduled as a tardy job in its designated tardy batch, but ends up being early when we reschedule job (i, j) into the last early batch for customer i in σ. Figure 5.2 depicts the schedule σ generated in step 5 by scheduling job (i, n_i) as early at the end of the current batch followed by rescheduling all tardy jobs from the designated tardy batch by adding them immediately after (i, n_i) into the current batch. (The schedule generated in step 6 would look similar, except job (i, n_i) would be the first job starting a new current batch.).

Since job (i,j) became early after rescheduling, we have $t_1 \leq d_{(i,j)} \leq d_{(i,n_i)}$, where t_1 is the completion time of the last batch for customer i. Let (i,j_2) be the first job in the batch whose due date is at least $d_{(i,j)}$. Move job (i,j) to the left in the batch by inserting it immediately before (i,j_2), and repeat this forward insertion operation for every job (i,j) that became early in σ. The Gantt chart of the resulting schedule is depicted in Figure 5.3. Job (i,j_1) denotes the last job in the preceding batch for customer i (if any). If $d_{(i,j_1)} > d_{(i,j)}$ for one of the moved jobs (i,j), then move (i,j_1) to immediately follow each such (i,j) in the last batch. The resulting schedule is depicted in Figure 5.4.

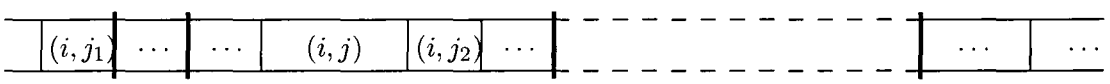


Figure 5.3: A full schedule with $d_{(i,j_1)} \leq d_{(i,j)}$.

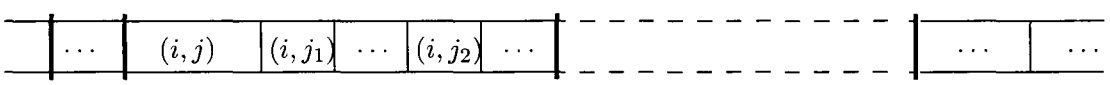


Figure 5.4: A full schedule with $d_{(i,j_1)} > d_{(i,j)}$.

If there are any other jobs in the preceding batches for customer i whose due date is greater than such a $d_{(i,j)}$, repeat this operation for these jobs. If we again consider only the early jobs in the resulting schedule, they are in EDD order after these operations. Thus the resulting early job set with this partitioning must have been generated some time during the execution of steps 1 to 4, since Algorithm CH5-A1 considers every early schedule.

After performing the above transformations for every job (i,j) that may have become early in schedule σ, we obtain a schedule σ' whose early job set is exactly the same as the early job set of the schedule σ produced by the rescheduling steps 5 or 6. Moreover, the early jobs are in EDD order in this transformed schedule σ' after the transformations. Therefore, its early job set with the same partitioning

must have been generated and considered during the execution of steps 1 to 4, since Algorithm CH5-A1 considers every early schedule. Furthermore, notice that job (i, n_i) is contained in this early job set, since it was early in σ, and the set of tardy jobs in the schedule σ' (implicitly scheduled in its designated tardy batch) would be the same as the set of tardy jobs in σ. Therefore, right after Algorithm CH5-A1 generated σ' (in one of steps 1 to 4), it would have considered rescheduling all the tardy jobs for customer i from its designated tardy batch by putting them right after (i, n_i) in σ'. Let us call this last schedule with the tardy jobs attached right after (i, n_i) schedule σ''. Notice that every state variable for σ'', except the objective value v'', must have the same value as the corresponding state variable for σ. Furthermore, the v'' value contains the tardiness penalties exactly for the tardy jobs in σ'' (and σ) and thus correctly represents the objective value for the corresponding state and $v > v''$. ■

Lemma 5.3.1 means that even if Algorithm CH5-A1 generates a partial schedule σ with corresponding state $(\mathcal{J}^k, \mathcal{H}, t, h, d, v)$ such that v does not represent the correct tardiness cost of the schedule, this will create no problem because in this case, it will always generate an alternative schedule σ'' with lower and correct cost. Since in Algorithm CH5-A1 we choose the state with smallest v from $\mathcal{S}^{(n)}$ as the final solution, the algorithm will correctly identify an optimal schedule at the end.

In order to study the complexity of the above algorithm, we introduce some further notations. Let $P = \sum_{(i,k)\in J} p_{(i,k)}$ be the total processing time of all jobs, $Q = \sum_{i\in M} n_i q_i$ an upper bound on the total delivery cost for all schedules, $S = \sum_{i\in M} n_i s_i$ an upper bound on the total batch-setup time for all schedules, and $d_{\max} = \max_{(i,k)\in J}\{d_{(i,k)}\}$ the maximum due date over all jobs. Recall that $W = \sum_{(i,k)\in J} w_{(i,k)}$.

Theorem 5.3.1 *Algorithm CH5-A1 runs in* $O(n^{m+1}P^m[\min\{d_{\max}, P+S\}][W+Q])$ *time and space and returns an optimal schedule for the* $1|s_i|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ *problem. This shows that the problem is* $\mathcal{NP}$*-hard only in the ordinary sense.*

Proof. Let us calculate the total number of possible different states. Each state is represented by six entries: $\{\mathcal{J}^k, \mathcal{H}, t, h, d, v\}$, $k = 0, 1, \cdots, n$. We know that $\mathcal{J}^k = (k_1, k_2, \cdots, k_m)$ with $k = \sum_{i=1}^{i=m} k_i$. Any $k_i \in \mathcal{J}^k$ can take at most $k+1$ different values,

$\{0, 1, \cdots, k\}$, and the first $(m-1)$ k_i values and k also determine k_m. Therefore, we have at most $(k+1)^{m-1}$ different vectors $\mathcal{J}^k$. Because the maximum k value is n, the upper bound for the number of $\mathcal{J}^k$ is $(n+1)^{m-1}$ for all k. We also have $\mathcal{H} = (\tau_1, \tau_2, \cdots, \tau_m)$, and any $\tau_i \in \mathcal{H}$ can take at most $p_i + 1$ different values from 0 to $p_i = \sum_{(i,k)\in J_i} p_{(i,k)}$. Therefore, we can have at most $\prod_{i=1}^{i=m}(p_i+1)$ different vectors $\mathcal{H}$. Because P is an upper bound of p_i for all i, the total number of different $\mathcal{H}$ vectors is bounded by $(P+1)^m$. As both h and d carry information for the job that is scheduled as the first job in the current early batch, we have at most n different combinations. Since t is the total processing and setup time of the early batches, its value must be between 0 to $\min\{d_{\max}, P+S\}$. Since v is the objective value, it is upper-bounded by $[W+Q]$. In summary, at the beginning of the k-th iteration there are at most $(n+1)^{m-1}(P+1)^m n[\min\{d_{\max}, P+S\}][W+Q]$ different states that need to be considered. For each state, there are at most m candidate jobs to be considered as the next job. For each job we generate at most six different new states in steps 1-6. Therefore, the time complexity of each iteration is $O(m(n+1)^{m-1}(P+1)^m n[\min\{d_{\max}, P+S\}][W+Q])$, which is $O(n^m P^m[\min\{d_{\max}, P+S\}][W+Q])$ for fixed m. Since we have n iterations, the overall running time is $O(n^{m+1}P^m[\min\{d_{\max}, P+S\}][W+Q])$. ■

5.4 Restricted Problem

The difficulty of converting Algorithm CH5-A1 into an FPTAS is how to eliminate the pseudo-polynomial term $\min\{d_{\max}, P+S\}$ when implementing the general framework of obtaining an FPTAS from a dynamic programming algorithm suggested by Sahni [1976]. We know that the term $\min\{d_{\max}, P+S\}$ records the length of the designated tardy jobs of each customer and it is used when rescheduling these designated tardy jobs in Algorithm CH5-A1.

Consider a restricted version of the problem, where all tardy jobs (if any) have to be delivered in separate batches, denoted by $1|s_i,\text{SEP}|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$. We are able to develop another pseudo-polynomial algorithm for it, which does not need

to record the length of the designated tardy jobs for any customer.

5.4.1 Pseudo-polynomial Algorithm

The following pseudo-polynomial algorithm for solving the $1|s_i,\text{SEP}|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem is based on Propositions 5.2.1, 5.2.2, 5.2.3, 5.2.4 and 5.2.5. The state space used to describe partial schedules contains state $\mathcal{L} = (l_1, ..., l_m)$, which is a zero-one vector signalling whether there is already a designated tardy batch for customer i ($l_i = 1$) or not ($l_i = 0$) and other five entries, $\{\mathcal{J}^k, t_2, h, d, v_2\}$ which are the same as defined in Section 5.3. Notice that we do not include entry $\mathcal{H} = (\tau_1, \tau_2, \cdots, \tau_m)$, which causes the term $\min\{d_{\max}, P + S\}$ in the complexity of Algorithm CH5-A1.

Before we describe Algorithm CH5-A2 in detail, let us consider how we can reduce the state space. Consider any two states $(\mathcal{J}^k, \mathcal{L}, t_1, h, d, v_1)$ and $(\mathcal{J}^k, \mathcal{L}, t_2, h, d, v_2)$. Without loss of generality, let $t_1 \leq t_2$. If $v_1 \leq v_2$, we can eliminate the second state because any later states which could be generated from the second state can not lead to better v value than the value of similar states generated from the first state. This validates the following elimination rule, and a similar argument could be used to justify the second remark.

Remark 5.4.1 *For any two states with the same entries $(\mathcal{J}^k, \mathcal{L}, t, h, d, \cdot)$, we can eliminate the state with larger v.*

Remark 5.4.2 *For any two states with the same entries $(\mathcal{J}^k, \mathcal{L}, \cdot, h, d, v)$, we can eliminate the state with larger t.*

Algorithm CH5-A2 follows the same pattern as Algorithm CH5-A1 and starts from the empty state where no job has been scheduled yet. Then Algorithm CH5-A2 recursively generates the states for the partial schedules on early jobs and at the same time designates the remaining jobs in a state corresponding to $\mathcal{J}^k$ to be tardy. The tardy jobs, however, will be processed and delivered separately later. We also assume without loss of generality, that jobs in each job set J_i, $i \in M$, are numbered in EDD order.

Algorithm CH5-A2

[Initialization] Set $\mathcal{S}^{(0)} = \{(\mathcal{J}^0, \mathcal{L}, 0, 0, 0, 0)\}$, $\mathcal{L} = (0, ..., 0)$ and $\mathcal{S}^{(k)} = \emptyset$, $k = 1, ..., n$.

[Generation] Generate set $\mathcal{S}^{(k)}$ from $\mathcal{S}^{(k-1)}$.

For $k = 1$ **to** n

Set $\mathcal{T} = \emptyset$;

For each $(\mathcal{J}^{k-1}, \mathcal{L}, t, h, d, v) \in \mathcal{S}^{(k-1)}$

For each $i \in M$ with $k_i < n_i$, set $k_i' = k_i + 1$ and $\mathcal{J}^k = (k_1, \cdots, k_i', \cdots, k_m)$

1. If $i = h$ and $t + p_{(i,k_i')} \leq d$, then set $\mathcal{T} \leftarrow \mathcal{T} \cup (\mathcal{J}^k, \mathcal{L}, t + p_{(i,k_i')}, h, d, v)$ */*Schedule job (i, k_i') early in the current early batch;*
2. If $t + p_{(i,k_i')} + s_i \leq d_{(i,k_i')}$, then set $\mathcal{T} \leftarrow \mathcal{T} \cup (\mathcal{J}^k, \mathcal{L}, t + p_{(i,k_i')} + s_i, i, d_{(i,k_i')}, v + q_i)$ */*Schedule job (i, k_i') early starting a new early batch.*
3. If $l_i = 1$, then set $\mathcal{T} \leftarrow \mathcal{T} \cup (\mathcal{J}^k, \mathcal{L}, t, h, d, v + w_{(i,k_i')})$ */*Schedule job (i, k_i') tardy in the existing designated tardy batch for customer i;*
4. If $l_i = 0$, then set $l_i = 1$ and $\mathcal{T} \leftarrow \mathcal{T} \cup (\mathcal{J}^k, \mathcal{L}, t, h, d, v + w_{(i,k_i')} + q_i)$ */*Schedule job (i, k_i') tardy starting a designated tardy batch for customer i.*

Endfor

[Elimination] Update set $\mathcal{S}^{(k)}$.

1. For any two states $(\mathcal{J}^k, \mathcal{L}, t, h, d, v)$ and $(\mathcal{J}^k, \mathcal{L}, t, h, d, v')$ with $v \leq v'$, eliminate the one with v' from set $\mathcal{T}$;
2. For any two states $(\mathcal{J}^k, \mathcal{L}, t, h, d, v)$ and $(\mathcal{J}^k, \mathcal{L}, t', h, d, v)$ with $t \leq t'$, eliminate the one with t' from set $\mathcal{T}$;
3. Set $\mathcal{S}^{(k)} = \mathcal{T}$.

Endfor

Endfor

[Result]: Select the state with the smallest v from set $\mathcal{S}^{(n)}$ and trace back to obtain an optimal schedule.

Theorem 5.4.1 *For the* $1|s_i,SEP|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ *problem, Algorithm CH5-A2 finds an optimal schedule in* $O(n^{m+1}\min\{d_{\max}, P+S, W+Q\})$ *time, where* $W = \sum_{(i,k)\in J} w_{(i,k)}$, $Q = \sum_{i\in M} n_i q_i$, $P = \sum_{(i,k)\in J} p_{(i,k)}$, $S = \sum_{i\in M} n_i s_i$ *and* $d_{\max} = \max_{(i,k)\in J}\{d_{(i,k)}\}$.

Proof. The correctness of Algorithm CH5-A2 follows from the previous propositions and discussion. Notice that the procedure [Generation] is the most time-consuming part. At the beginning of iteration k, the total number of possible states $(\mathcal{J}^{k-1}, \mathcal{L}, t, h, d, v)$ in $\mathcal{S}^{(k-1)}$ is upper-bounded by the following: there are at most k^{m-1} different $\mathcal{J}^k$ for fixed k, at most 2^m different $\mathcal{L}$ vectors, at most m values of h and n values of d, at most $\min\{d_{\max}, P+S\}$ possible values for t and $W+Q$ different values for v. Because of the elimination rules, $\min\{d_n, P+S, W+Q\}$ is an upper bound for the number of different combinations of t and v. Thus the total number of different states at the beginning of each iteration is $O(k^{m-1}2^m mn\min\{d_{\max}, P+S, W+Q\})$. In each iteration k, there are at most three new states generated from each state in $\mathcal{S}^{(k-1)}$ for each candidate job, and there are at most m candidate jobs. Since there are n iterations, the [Generation] procedure could indeed be done in $O(3nk^{m-1}2^m m^2 n\min\{d_{\max}, P+S, W+Q\})$ time, which is $O(n^{m+1}\min\{d_{\max}, P+S, W+Q\})$. ■

Corollary 5.4.1 *For the* $1|s_i,SEP|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ *problem, if all jobs have equal tardiness penalties, i.e.,* $w_{(i,k)} = w > 0$, $\forall (i,k) \in J$ *and all customers have equal batch-delivery costs, i.e.,* $q_i = q > 0$, $\forall i \in M$, *then Algorithm CH5-A2 provides an optimal solution for it in* $O(n^{m+3})$ *time.*

Proof. For any state, v is the sum of two different cost components: the delivery costs from the set $\{q, 2q, ..., nq\}$ and the weighted number of tardy jobs from $\{0, w, ..., nw\}$.

Therefore, v can take at most $n(n+1)$ different values, and $W+Q$ turns into $O(n^2)$. Thus the time complexity becomes $O(n^{m+1}\min\{d_{\max}, P+S, W+Q\}) = O(n^{m+3})$. ■

Corollary 5.4.2 *For the* $1|s_i,SEP|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ *problem, if all jobs have equal processing times, i.e.,* $p_{(i,k)} = p > 0$, $\forall (i,k) \in J$ *and all customers have equal batch-setup times, i.e.,* $s_i = s > 0$, $\forall i \in M$, *then Algorithm CH5-A2 provides an optimal solution for it in* $O(n^{m+3})$ *time.*

Proof. For any state, t is the sum of two different time components: the setup times from $\{s, ..., ns\}$ and the processing times from $\{0, p, ..., np\}$. Thus, t can take at most $n(n+1)$ different values, and $P+S$ turns into $O(n^2)$. Thus the time complexity becomes $O(n^{m+1}\min\{d_{\max}, P+S, W+Q\}) = O(n^{m+3})$. ■

5.4.2 Fully Polynomial Time Approximation Scheme

In this subsection, we will convert Algorithm CH5-A2 into an FPTAS using the static interval partitioning approach suggested by Sahni [1976]. To be able to implement this approach fast, we need a pair of tight bounds such that the upper bound is a constant multiple of the lower bound. In what follows, we first develop two new algorithms to determine an initial pair of upper and lower bounds. After this, we present two other algorithms to narrow the initial bounds into a pair of tight bounds. Finally, using the tight bounds, we are able to implement the $(1+\varepsilon)$-approximation algorithm in fully polynomial time for any given $\varepsilon > 0$.

Step I: Initial Bounds

To obtain a pair of initial bounds, we need to construct an auxiliary problem using the same data, where we assume $q_i = 0$, $\forall i \in M$, and the goal is to minimize the maximum weight of tardy jobs with batch-setup time s_i, $i \in M$, denoted by $1|s_i|\max w_{(i,k)}U_{(i,k)}$. Since any feasible schedule for the $1|s_i|\max w_{(i,k)}U_{(i,k)}$ problem is also feasible for the $1|s_i,\text{SEP}|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem and vice versa, the optimal solution value

for the $1|s_i|\max w_{(i,k)}U_{(i,k)}$ problem is a lower bound for the minimum cost of the optimal solution value for the $1|s_i,\text{SEP}|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem.

Temporarily assume, without loss of generality, that all jobs are reindexed in the smallest-weight-first order, $w_{[1]} \leq w_{[2]} \leq ... \leq w_{[n]}$, where job $[k]$ is the job with the overall k-th smallest weight, regardless which customer it belongs to. It is easy to see that if the optimal solution for the $1|s_i|\max w_{(i,k)}U_{(i,k)}$ problem is $w_{[k^*]}$ for some k^*, then there must exist a schedule in which jobs $[k^*+1],...,[n]$ can be delivered early, and there is no schedule in which all of jobs $[k^*],[k^*+1],...,[n]$ can be delivered early. Based on this, to solve the $1|s_i|\max w_{(i,k)}U_{(i,k)}$ problem, we first propose a feasibility checking algorithm, Algorithm CH5-A3(k) that can report whether there is a feasible schedule for the $1|s_i|\max w_{(i,k)}U_{(i,k)}$ problem in which the last $n-k$ jobs (taken from the smallest-weight-first order) are early. Then we can determine k^* by repeatedly using Algorithm CH5-A3(k) in a binary search of the range $[0,n]$.

Notice that all propositions for early batches discussed before are also applicable to the $1|s_i|\max w_{(i,k)}U_{(i,k)}$ problem. Thus if there is a feasible schedule in which all jobs in $\Phi(k) = \{[k+1],...,[n]\}$ are early, then there is also one in which they are in EDD order. Therefore, Algorithm CH5-A3(k) is applied to a representation of the job set $\Phi(k)$ in which jobs are identified by which customer they belong to and are re-sorted into EDD order: Assume that the job set $\Phi(k) = \{[k+1],...,[n]\}$ is further grouped into m subsets $\Phi_i(k)$ with respect to the m customers, $i \in M$, and each $\Phi_i(k) = \{(i,1),...,(i,\theta_i)\}$ contains its θ_i jobs in EDD order, where $\sum_{i\in M}\theta_i = n-k$. (It is possible that $\Phi_i(k) = \emptyset$, i.e., $\theta_i = 0$, for some $i \in M$.) Let $(\mathcal{I}^f, h, d, t)$ be the state representing a partial schedule with makespan t on the combined job set $\{(i,1),...,(i,f_i)\}$, $i \in M$ where $f_i \leq \theta_i$, $f = \sum_{i=1}^m f_i$ and $\mathcal{I}^f = (f_1,...,f_m)$, and similarly to our previous notation, the job with batch-due date d in the current batch belongs to customer h. Algorithm CH5-A3(k) starts from the empty schedule $(\mathcal{I}^0,0,0,0)$, where $\mathcal{I}^0 = (0,...,0)$. The set of feasible partial schedules at stage $f \in \{1,2,...,n-k\}$ is denoted by $\mathcal{R}^{(f)}$, and initially we have $\mathcal{R}^{(f)} = \emptyset$. Note that all job sets scheduled in a partial schedule belonging to $\mathcal{R}^{(f)}$ have the common cardi-

nality f, but these sets differ in their defining $\mathcal{I}^f = (f_1, f_2, ..., f_m)$ vectors. For each setting $(\mathcal{I}^f, h, d, \cdot)$, we will carry only the partial feasible schedule with the minimum t value identified by Algorithm CH5-A3(k) so far. Thus for each $(\mathcal{I}^f, h, d, \cdot)$, only one state $(\mathcal{I}^f, h, d, t)$ is stored at any time. Initially, we start with $(\mathcal{I}^f, h, d, \infty)$.

Algorithm CH5-A3(k)

[Initialization] Set $\mathcal{R}^{(0)} = \{(\mathcal{I}^0, 0, 0, 0)\}$, where $\mathcal{I}^0 = (0, ..., 0)$.

[Subgrouping] For the given k value, construct $\Phi_i(k) = \{(i, 1), ..., (i, \theta_i)\}$ from job set $\Phi(k) = \{[k+1], ..., [n]\}$, where $\sum_{i \in M} \theta_i = n - k$, $d_{(i,1)} \leq ... \leq d_{(i,\theta_i)}$, $i \in M$.

[FeasibilityChecking] Schedule one more job into the partial schedule.

For $f = 1$ **to** $n - k$, generate set $\mathcal{R}^{(f)}$ from $\mathcal{R}^{(f-1)}$

Initialize $t = \infty$ for all possible $\mathcal{I}^f$, h and d, i.e., add $(\mathcal{I}^f, h, d, \infty)$ to $\mathcal{R}^{(f)}$.

For each $(\mathcal{I}^{f-1}, h, d, t) \in \mathcal{R}^{(f-1)}$ with $\mathcal{I}^{f-1} = (f_1, f_2, ..., f_m)$

For $i = 1$ **to** m, if $f_i + 1 \leq \theta_i$, then

1. Set $f_i = f_i + 1$.
2. If $i = h$, $t + p_{(i,f_i)} \leq d$ and $t + p_{(i,f_i)} < \tau$, where τ is the best value found so far for the state $(\mathcal{I}^f, h, d, \ )$ stored as $(\mathcal{I}^f, h, d, \tau)$, then set $\tau = t + p_{(i,f_i)}$ for the state. */*Job (i, f_i) can be added to the current batch and the schedule's total processing time is less than τ.*

2. Consider the τ in the state $(\mathcal{I}^f, i, d_{(i,f_i)}, \tau)$: If $t + p_{(i,f_i)} + s_i \leq d_{(i,f_i)}$ and $t + p_{(i,f_i)} + s_i < \tau$, then set $\tau = t + p_{(i,f_i)} + s_i$ for the state. */*Job (i, f_i)can be added starting a new batch and the schedule's total processing time is less than the best value τ obtained so far.*

Endfor

Endfor

Endfor

[Result]: If there is no state $(\mathcal{I}^{n-k}, h, d, \tau)$ in $\mathcal{R}^{(n-k)}$ with $\tau < \infty$, then no feasible schedule exists; Otherwise, there exists a feasible schedule in which all jobs in $\Phi(k) = \{[k+1], ..., [n]\}$ are early.

Theorem 5.4.2 *Algorithm CH5-A3(k) correctly reports whether there exists a feasible schedule on $\Phi(k)$ and it requires $O(n^{m+1})$ time.*

Proof. In [Subgrouping], we need to construct m EDD-ordered subsequences of the $n-k$ jobs in $\Phi(k)$. This can be done in $O(m(n-k)\log(n-k))$ time. At the beginning of each iteration f of [FeasibilityChecking], there are at most $(n-k)^m$ states in $\mathcal{R}^{(f-1)}$, because there can be at most $(n-k)^{m-1}$ different $\mathcal{I}^{f-1}$ vectors, each vector may appear with at most $(n-k)$ different (h, d) combinations, and Algorithm CH5-A3(k) only keeps one partial schedule for each $(\mathcal{I}^{f-1}, h, d, \cdot)$, the one with the smallest makespan τ. The internal loops i of [FeasibilityChecking] require $O(m)$ time. Overall, there are $(n-k)$ [FeasibilityChecking] iterations. In summary, this means that Algorithm CH5-A3(k) runs in $O(m(n-k)^{m+1})$ time, which is $O(n^{m+1})$. ■

Next, we present the following Algorithm CH5-A4 that finds an optimal solution for $1|s_i|\max w_{(i,k)}U_{(i,k)}$. Suppose that all jobs are indexed in smallest-weight-first order, i.e., $w_{[1]} \leq w_{[2]} \leq ... \leq w_{[n]}$. Thus the optimal result is the smallest k for which Algorithm CH5-A3(k) reports a feasible schedule. It means that we can schedule jobs $\{[k], [k+1], ..., [n]\}$ early in the $1|s_i|\max w_{(i,k)}U_{(i,k)}$ problem, but we can not schedule all of $\{[k-1], [k], ..., [n]\}$ early. We will apply Algorithm CH5-A3(k) in a standard binary search to find this smallest k, denoted by k^* in Algorithm CH5-A4.

Algorithm CH5-A4

[Initialization] Set $k_1 = 0$, $k_2 = n$ and $k^* = 0$.

While $k_1 < k_2$, **do**

1. Set $k = \lceil (k_1 + k_2)/2 \rceil$ and run Algorithm CH5-A3(k);

2. If it reports that no feasible schedule exists, then set $k_1 = k$;
3. If it reports a feasible schedule, then set $k^* = k$ and $k_2 = k$.

Endwhile

[Result] If $k^* = 0$, then $w^* = 0$; Otherwise, $w^* = w_{[k^*]}$.

Theorem 5.4.3 *Algorithm CH5-A4 runs in $O(n^{m+1} \log n)$ time and finds the optimal solution value for the $1|s_i| \max w_{(i,k)}U_{(i,k)}$ problem.*

Proof. Note that Algorithm CH5-A4 needs to call Algorithm CH5-A3(k) at most $\log n$ times to find k^*. ■

The next lemma shows that Algorithm CH5-A4 provides initial lower and upper bounds for the $1|s_i,\text{SEP}|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem.

Lemma 5.4.1 *If v_1^* is the optimal solution value for the $1|s_i,SEP|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem, then $v' \leq v_1^* \leq nv'$, where $v' = w^* + \sum_{i \in M} q_i$ and w^* is the optimal solution value for the $1|s_i| \max w_{(i,k)}U_{(i,k)}$ problem.*

Proof. By the definitions of the $1|s_i,\text{SEP}|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem and the $1|s_i| \max w_{(i,k)}U_{(i,k)}$ problem, we know that any feasible schedule of the latter one is also feasible for the former one and vice versa. Therefore, the total weighted number of tardy jobs in an optimal schedule for the $1|s_i,\text{SEP}|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem is at least w^*. Furthermore, there is at least one batch for each customer in any schedule for the $1|s_i,\text{SEP}|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem. Thus $v' = w^* + \sum_{i \in M} q_i$ is a lower bound for v_1^*. Since the optimal schedule for the $1|s_i| \max w_{(i,k)}U_{(i,k)}$ problem has at most k^* tardy jobs, the total weighted number of tardy jobs is at most k^*w^* in this schedule. Furthermore, it clearly cannot have more than n deliveries. Therefore, $k^*w^* + n\sum_{i \in M} q_i \leq n(w^* + \sum_{i \in M} q_i) = nv'$ is an upper bound for v_1^*. ■

Step II: Tight Bounds

In order to narrow the gap between the lower and upper bounds given in the previous lemma so that the ratio of the resulting upper and lower bound is itself bounded by

a constant. We use the same approach as the strategy employed in Chapter 4 for the $1|s|\sum w_iU_i + bq$ problem.

Algorithm CH5-A5(u, ε) is very similar to Algorithm CH5-A2 with a certain variation of the [Elimination] and [Result] procedures. Assume that the jobs have been sorted into EDD order in each J_i, $\forall i \in M$.

Algorithm CH5-A5(u, ε)

[Initialization] Do the same as in Algorithm CH5-A2.

[Partitioning] Partition the interval $[0, u]$ into $\lceil n/\varepsilon \rceil$ equal intervals of size $u\varepsilon/n$, with the last one being possibly smaller.

[Generation] Generate set $\mathcal{S}^{(k)}$ from $\mathcal{S}^{(k-1)}$.

For $k = 1$ **to** n

Set $\mathcal{T} = \emptyset$;

For each $(\mathcal{J}^{k-1}, \mathcal{L}, t, h, d, v) \in \mathcal{S}^{(k-1)}$

For each $i \in M$ with $k_i < n_i$, set $k_i' = k_i + 1$ and $\mathcal{J}^k = (k_1, \cdots, k_i', \cdots, k_m)$

Do the same as in Algorithm CH5-A2.

Endfor

Endfor

[Elimination] Update set $\mathcal{S}^{(k)}$.

1. Eliminate any state $(\mathcal{J}^k, \mathcal{L}, t, h, d, v)$ if $v > u$.
2. If more than one state has a v value that falls into the same subinterval of $[0, u]$, then discard all but one of these states, keeping only the representative state with the smallest t coordinate for each interval.
3. For any two states $(\mathcal{J}^k, \mathcal{L}, t, h, d, v)$ and $(\mathcal{J}^k, \mathcal{L}, t, h, d, v')$ with $v \leq v'$, eliminate the one with v' from set $\mathcal{T}$;
4. Set $\mathcal{S}^{(k)} = \mathcal{T}$.

Endfor

[Result] If $\mathcal{S}^{(n)} = \emptyset$, $v_1^* > (1-\varepsilon)u$; Otherwise $v_1^* \leq u$.

Theorem 5.4.4 *If Algorithm CH5-A5(u, ε) reports $S^{(n)} = \emptyset$, then $v_1^* > (1-\varepsilon)u$; otherwise $v_1^* \leq u$. The time complexity of Algorithm CH5-A5(u, ε) is $O(n^{m+2}/\varepsilon)$ for arbitrary $\varepsilon > 0$.*

Proof. Similarly to Theorem 4.4.3, if $\mathcal{S}^{(n)}$ is not empty, then there is at least one state $(\mathcal{J}^n, \mathcal{L}, t, h, d, v)$ with v in some subinterval of $[0, u]$ that has not been eliminated. Therefore, we have $v_1^* \leq v \leq u$. If $\mathcal{S}^{(n)} = \emptyset$, by [Partitioning] and [Elimination], the error introduced in one iteration is at most $\varepsilon u/n$, the overall error is at most $n(\varepsilon u/n) = \varepsilon u$, i.e., $v \geq (1-\varepsilon)u$. Thus $v_1^* > (1-\varepsilon)u$.

Notice that $|\mathcal{S}^{(k)}| \leq \lceil n^{m+1}/\varepsilon \rceil$ for $k = 1, 2, ..., n$. Since all operations on a single state can be performed in $O(m)$ time and there are n iterations in the [Generation] procedure, the overall complexity is $O(n^{m+2}/\varepsilon)$. ■

Again, we consider to use $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ to narrow the bounds $[v', nv']$. Let $a_1 = v'$, $a_2 = nv'$, $\omega = u$ and $\mathcal{P}$ be the $1|s_i,\text{SEP}|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem. Then Algorithm CH5-A5$(u, 1/3)$ can be used as $\mathcal{A}(\omega, \mathcal{P})$. Since Algorithm CH5-A5$(u, 1/3)$ runs in $O(n^{m+2})$ time, $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ reports ξ in $O(n^{m+2}\log\log n)$ time, which implies a pair of tight bounds $[\xi, 3\xi]$. In this implementation, we refer to $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ as Algorithm CH5-A6.

Corollary 5.4.3 *Algorithm CH5-A6 can narrow the bounds $[v', nv']$ into $[\bar{v}, 3\bar{v}]$ in $O(n^{m+2}\log\log n)$ time by setting $\bar{v} = \xi$.*

Step III: Approximation

For any given $\varepsilon > 0$, using the bounds $\bar{v} \leq v_1^* \leq 3\bar{v}$ obtained by Algorithm CH5-A6, we run a slightly changed version of Algorithm CH5-A5(u, ε), called Algorithm CH5-A7, with $u = (1+\varepsilon/3)3\bar{v}$: The only difference is that in the [Partitioning] step we partition $[0, u] = [0, (1+\varepsilon/3)3\bar{v}]$ into $n\lceil 3/\varepsilon + 1\rceil$ intervals of size at most $\varepsilon\bar{v}/n$, so that

the cumulative error over n iterations will be no more than $\varepsilon\bar{v}$. Since we know that the problem has an optimal solution value $v_1^* \leq 3\bar{v}$, the algorithm will find a solution v for the $1|s_i,\text{SEP}|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem such that $v \leq 3\bar{v} + \varepsilon\bar{v} = u$, which means that the algorithm will never end with an empty $\mathcal{T}$. Furthermore, whichever subinterval of $[\bar{v}, 3\bar{v}]$ v_1^* falls into, the algorithm will generate an approximate solution v with at most $\varepsilon\bar{v}$ error away from it, i.e., $v \leq v_1^* + \varepsilon\bar{v} \leq (1+\varepsilon)v_1^*$. Therefore, we have the following corollary.

Corollary 5.4.4 *For any given $\varepsilon > 0$, Algorithm CH5-A7 finds a $(1+\varepsilon)$-approximate solution for the $1|s_i,SEP|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem in $O(n^{m+2}/\varepsilon)$ time.*

Now we are ready to present the final Algorithm CH5-A8, which combines into an FPTAS the previous algorithms as subroutines.

Algorithm CH5-A8

[InitialBounds]: Run Algorithm CH5-A4 and set $v' = w^* + \sum_{i\in M} q_i$;

[TightBounds]: Run Algorithm CH5-A6 and obtain: $\bar{v} \leq v_1^* \leq 3\bar{v}$.

[Approximation]: Run Algorithm CH5-A7 and obtain an approximate schedule.

Theorem 5.4.5 *For the $1|s_i,SEP|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem, Algorithm CH5-A8 finds a $(1+\varepsilon)$-approximate solution in $O(n^{m+2}/\varepsilon + n^{m+2}\log\log n)$ time.*

Proof. The complexity and correctness follow from the results for the component algorithms. ■

5.5 Approximation Algorithm

In this section, we show that the approximate solution found by Algorithm CH5-A8 for the $1|s_i,\text{SEP}|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem is also near-optimal for the original $1|s_i|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem. Note that any feasible schedule for the former

one is also a feasible schedule for the latter one and its objective value is the same for both versions. Thus if σ is any batching schedule with late jobs delivered separately, then $v_1(\sigma) = v(\sigma)$, where $v_1(\sigma)$ and $v(\sigma)$ are the objective value of σ for the $1|s_i,\text{SEP}|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ and $1|s_i|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem, respectively. Therefore, $v^* \leq v_1^*$. We can have three cases for the original problem:

1. If there is an optimal schedule in which all jobs are early, then this schedule is optimal for both problems, and $v_1^* = v^*$;

2. If there is an optimal schedule in which all tardy jobs are delivered separately, then this schedule is optimal for both problems, and $v_1^* = v^*$;

3. If in any optimal schedule for the original $1|s_i|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem, there is at least one customer whose tardy jobs must be delivered together with some early jobs, then an optimal schedule for the $1|s_i,\text{SEP}|\sum w_{(i,k)}U_{(i,k)}+\sum b_i q_i$ problem is not optimal for the $1|s_i|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem. However, since an optimal schedule for the $1|s_i,\text{SEP}|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem has at most one separate tardy batch delivered for each customer, we have $v_1^* \leq v^* + \sum_{i=1}^m q_i$.

In summary, we always have $v_1^* \leq v^* + \sum_{i=1}^m q_i$ in all cases. Let us define $\delta = \frac{w^*}{\sum_{i=1}^m q_i}$.

Corollary 5.5.1 *For any $\varepsilon > 0$, Algorithm CH5-A8 is $\rho = (2+\delta+2\varepsilon+\delta\varepsilon)/(1+\delta)$ approximation algorithm for the $1|s_i|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem with time complexity $O(n^{m+2}\log n + n^{m+2}/\varepsilon)$, where $\delta = \frac{w^*}{\sum_{i=1}^m q_i}$.*

Proof. Let σ_ε be a schedule produced by Algorithm CH5-A8 for $\varepsilon > 0$. By Theorem 5.4.5, $v_1(\sigma_\varepsilon) \leq (1+\varepsilon)v_1^*$. Furthermore,

$$(1+\delta)\sum_{i=1}^m q_i = w^* + \sum_{i=1}^m q_i \leq v^* \leq v_1(\sigma_\varepsilon) \leq (1+\varepsilon)v_1^* \leq (1+\varepsilon)(v^* + \sum_{i=1}^m q_i).$$

This implies that

$$v_1(\sigma_\varepsilon) - v^* \leq \varepsilon v^* + (1+\varepsilon)\sum_{i=1}^{m} q_i \leq \varepsilon v^* + \frac{1+\varepsilon}{1+\delta} v^* = v^* \frac{1+2\varepsilon+\delta\varepsilon}{1+\delta}, \tag{5.5.1}$$

which means that

$$\rho = \frac{v_1(\sigma_\varepsilon)}{v^*} \leq 1 + \frac{1+2\varepsilon+\delta\varepsilon}{1+\delta} = \frac{2+\delta+2\varepsilon+\delta\varepsilon}{1+\delta}. \tag{5.5.2}$$

The complexity follows from the running times of the component algorithms. ■

Since $\delta > 0$ and ρ is monotone decreasing in δ, we have $\rho \leq 2 + 2\varepsilon$ in the worst case. However, as the following table shows, the quality of the approximation by Algorithm CH5-A8 improves quite rapidly as δ is increasing.

δ	1/2	1	5	10	100
ρ	$5(1+\varepsilon)/3$	$3(1+\varepsilon)/2$	$7(1+\varepsilon)/6$	$12(1+\varepsilon)/11$	$102(1+\varepsilon)/100$

5.6 Summary

In this chapter, we first proved that the $1|s_i|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem is $\mathcal{NP}$-hard only in the ordinary sense by proposing a pseudo-polynomial algorithm for it. In order to obtain an FPTAS, we then studied a restricted version of the original problem: the $1|s_i,\text{SEP}|\sum w_{(i,k)}U_{(i,k)} + \sum b_i q_i$ problem. For this problem, we proposed a simpler pseudo-polynomial algorithm and an FPTAS. Finally, we showed that the FPTAS for this restricted problem is also an approximation algorithm for the original problem with a relatively good approximation ratio.

Chapter 6

Models with a Common Assignable Due Date

In Chapter 4, we have studied the supply chain scheduling model with delivery costs for a single customer. In the next three chapters, we include due date assignment in it. In particular, this chapter deals with only CON problems, where CON means that a common due date is assigned to all jobs.

6.1 Introduction

Meeting due dates always is one of the important concerns in scheduling and supply chain management. In traditional machine scheduling, due dates are given and defined in advance. According to the paper [Slotnick and Sobel, 2005], however, the tardiness penalties in aerospace industries may be as high as one million dollars per day for suppliers of aircraft components. In this situation, suppliers might consider extending due dates to allow jobs to be on time. In order to keep the customers at the same time, the suppliers may have to offer reduced prices as trade-offs. This motivates us to include due date assignment and its costs in the model studied in Chapter 4. We

start this research by studying problems in which a common due date is assigned to all jobs.

This chapter is organized as follows. In Section 6.2, we define the problems in detail and provide $\mathcal{NP}$-hardness proofs. In Section 6.3, we propose a pseudo-polynomial algorithm and an FPTAS for the unconstrained problem. In Section 6.4, we further prove that the algorithms for the unconstrained problem are applicable for the time-constrained problem just with a slight variation. In Section 6.5, we develop a pseudo-polynomial algorithm and an FPTAS for the capacity-constrained problem. In Section 6.6, we make our final conclusions.

6.2 Preliminaries

Note that all the terminologies, assumptions and notations introduced in Chapters 1 and 3 are applied in this chapter. Our goal is to find a schedule which minimizes the sum of the due-date-assignment costs, the weighted number of tardy jobs and the batch-delivery costs, denoted by $1|s, A,\text{CON}|\sum \alpha_j R_j + \sum w_j U_j + bq$. We also call the $1|s, A,\text{CON}|\sum \alpha_j R_j + \sum w_j U_j + bq$ problem the *unconstrained* problem, if there are unlimited number of vehicles available for delivering jobs and the delivery capacity on a single trip is unlimited. If the customer requires a time period of at least T between any two consecutive deliveries, we construct a *time-constrained* problem, denoted by $1|s, T, A,\text{CON}|\sum \alpha_j R_j + \sum w_j U_j + bq$. If the delivery capacity on a single trip is B, we construct a *capacity-constrained* problem, denoted by $1|s, B, A,\text{CON}|\sum \alpha_j R_j + \sum w_j U_j + bq$. We now prove their $\mathcal{NP}$-hardness.

Theorem 6.2.1 *The $1|s, A, CON|\sum \alpha_j R_j + \sum w_j U_j + bq$ (unconstrained) problem is $\mathcal{NP}$-hard.*

Proof. Consider the knapsack problem shown below, which is well-known to be $\mathcal{NP}$-hard. :

$$\text{Maximize } \sum_{j\in\Omega} w_j, \text{ such that: } \sum_{j\in\Omega} p_j \leq A,\ \Omega \subseteq J. \tag{6.2.1}$$

Construct an instance of the unconstrained problem from this, in which $s = q = 0$, $\alpha_j \gg w_j$ and p_j be the processing time, $\forall j \in J$. This implies that the largest value for the assigned due date will be $D = A$ in any optimal schedule. Suppose φ is the set of early jobs in an optimal schedule for the above instance of the unconstrained problem. Then to minimize the $\sum w_j U_j$, we know that φ must maximize the tardiness penalties of early jobs with $\sum_{j\in\varphi} p_j \leq A$. Therefore, by setting $\Omega = \varphi$, we obtain an optimal solution for the above knapsack problem. We proved the equivalence and thus the unconstrained problem is $\mathcal{NP}$-hard. ■

If we consider $T = 1$ or $B = n$, then the constraints are vacuous in the time-constrained or the capacity-constrained problem, respectively. Thus both these problems are just unconstrained problems and we have the following two corollaries.

Corollary 6.2.1 *The* $1|s, T, A, CON|\sum \alpha_j R_j + \sum w_j U_j + bq$ *(time-constrained) problem is* $\mathcal{NP}$*-hard.*

Corollary 6.2.2 *The* $1|s, B, A, CON|\sum \alpha_j R_j + \sum w_j U_j + bq$ *(capacity-constrained) problem is* $\mathcal{NP}$*-hard.*

6.3 Unconstrained Problem

In this section, we study the unconstrained problem, $1|s, A, \text{CON}|\sum \alpha_j R_j + \sum w_j U_j + bq$. For this problem, we first develop a polynomial algorithm for the case of $A = 0$ and then we propose a pseudo-polynomial algorithm and an FPTAS for the case of $A > 0$. The following proposition observes the structure of optimal schedules in both cases.

Proposition 6.3.1 *There is an optimal schedule for the* $1|s, A, CON|\sum \alpha_j R_j + \sum w_j U_j + bq$ *problem, in which there are at most two batches.*

Proof. Suppose that there are more than two batches in an optimal schedule, say σ. Since all jobs share the same due date, there are at least two early batches or two

tardy batches. In either case, delivering all early (or tardy) batches together saves batch-delivery costs for a batch without adding any costs. Therefore, schedule σ is not optimal, a contradiction. ■

6.3.1 Zero Contracted Due Date

For the unconstrained problem, when $A = 0$, denoted by $1|s, A = 0,\text{CON}|\sum \alpha_j R_j + \sum w_j U_j + bq$, we have $R_j = D$, $\forall j \in J$. We can observe the following facts.

Proposition 6.3.2 *There is an optimal schedule for the* $1|s, A = 0, CON|\sum \alpha_j R_j + \sum w_j U_j + bq$ *problem, in which* D *is either defined as a delivery time or equal to* 0.

Proof. Suppose the optimal schedule has all jobs delivered as early jobs at $P + s$ in a single batch, then setting $D = P + s$ has the minimum cost. On the other hand, suppose the optimal schedule has all jobs delivered as tardy jobs at $P + s$ in a single batch, then setting $D = 0$ has the minimum cost. Finally, suppose the optimal schedule has two batches and the first one is delivered at t, then setting $D = t$ has the minimum cost. ■

Proposition 6.3.3 *Consider schedules which have two batches for the* $1|s, A = 0, CON|\sum \alpha_j R_j + \sum w_j U_j + bq$ *problem, then delivering jobs in* $J_E = \{j | p_j < \frac{w_j}{\sum_{j=1}^{n} \alpha_j}, j \in J\}$ *as early jobs at* $D = \sum_{j \in J_E} p_j + s$, *and the remaining jobs as tardy jobs at the end generates a schedule which has the minimum cost.*

Proof. The schedule specified in the above proposition has total cost

$$(\sum_{j \in J_E} p_j + s) \sum_{j=1}^{n} \alpha_j + \sum_{j \in J \setminus J_E} w_j + 2q,$$

which can be divided into four parts: $s \sum_{j=1}^{n} \alpha_j$, $2q$, $\sum_{j \in J_E} p_j \sum_{j=1}^{n} \alpha_j$ and $\sum_{j \in J \setminus J_E} w_j$. Without considering the first two parts which every two-batch schedule incurs, job j contributes either $p_j \sum_{j=1}^{n} \alpha_j$ or w_j to the total cost. Thus scheduling job $j \in J_E$ in the first batch saves costs if $p_j < \frac{w_j}{\sum_{j=1}^{n} \alpha_j}$. ■

The following Algorithm CH6-A1, which is based on Propositions 6.3.1, 6.3.2 and 6.3.3, solves the problem.

Algorithm CH6-A1

1. Set $v_t, v_e, v_d = \infty$ /*Initialization.*
2. Set $D = 0$ and $v_t = q + \sum_{j \in J} w_j$ /*Jobs are scheduled tardy in one batch.*
3. Set $D = P + s$ and $v_e = q + D \sum_{j=1}^{n} \alpha_j$ /*Jobs are scheduled early in one batch.*
4. Determine $J_E = \{j | p_j < \frac{w_j}{\sum_{j=1}^{n} \alpha_j}, j \in J\}$.
5. If $0 < \sum_{j \in J_E} p_j < P$, then set $D = \sum_{j \in J_E} p_j + s$ and $v_d = 2q + \sum_{j \in J \setminus J_E} w_j + D \sum_{j=1}^{n} \alpha_j$ /*Jobs are scheduled in two batches and the first batch is early.*
6. Find the optimal solution by $v^* = \min\{v_t, v_e, v_d\}$ as the optimum and obtain the corresponding schedule and the corresponding D.

Theorem 6.3.1 *Algorithm CH6-A1 solves the* $1|s, A = 0, CON| \sum \alpha_j R_j + \sum w_j U_j + bq$ *problem in polynomial time.*

Proof. All the above steps involve calculations, which can be finished in $O(n)$ time. ■

6.3.2 Positive Contracted Due Date

Let $1|s, A > 0, \text{CON}| \sum \alpha_j R_j + \sum w_j U_j + bq$ denote the unconstrained problem with $A > 0$. Now we develop one more proposition as follows.

Proposition 6.3.4 *For the* $1|s, A > 0, CON| \sum \alpha_j R_j + \sum w_j U_j + bq$ *problem, if a two-batch schedule has the first batch delivered at* t*, then setting* $D = \max\{t, A\}$ *has the minimum cost.*

Proof. Since all jobs share a common due date, in any optimal schedule with two batches, the first batch is an early batch and the other batch is a tardy batch. Suppose the first batch is delivered at t. To let the first batch be early, we have $D \geq t$. If $t > A$, then the best is to set $D = t$. If $t \leq A$, then set $D = A$. In summary, we will have $D = \max\{t, A\}$. ■

Pseudo-polynomial Algorithm

We know from Proposition 6.3.1 that there are at most two batches in any optimal schedule. Let us first discuss the following two patterns of the schedules, which have exactly two batches.

Pattern I: Consider schedules in which there are two deliveries and the first delivery happens at or before A. By the proof of Theorem 6.2.1, looking for the lowest cost schedule is equivalent to solving the following knapsack problem, KP1(x):

$$\text{Maximize } \sum_{j\in\Omega} w_j, \text{ such that: } \sum_{j\in\Omega} p_j \leq x - s, \Omega \subset J, \tag{6.3.2}$$

where $x = A$. Then the resulting schedule is to deliver jobs in Ω at $\sum_{j\in\Omega} p_j + s \leq A$ and the remaining jobs at the end, at time $P+2s$. Therefore, if we set $D = A$, then the due-date-assignment cost is zero and the cost of the schedule is $v_I = \sum_{j\in J\setminus\Omega} w_j + 2q$.

Pattern II: Consider schedules in which there are two deliveries, and jobs in $\Phi \subset J$ are delivered in this first batch at $t > A$ as early jobs, where $t = \sum_{j\in\Phi} p_j + s$. Then, by Proposition 6.3.4, to obtain the lowest cost schedule, we set $D = \sum_{j\in\Phi} p_j + s$ and the total cost is

$$\begin{aligned} v_{II} &= (\sum_{k\in\Phi} p_k + s - A)\sum_{j\in J} \alpha_j + \sum_{j\in J\setminus\Phi} w_j + 2q \\ &= (s - A)\sum_{j\in J} \alpha_j + 2q + \sum_{j\in J} p_j \sum_{k\in J} \alpha_k - \sum_{j\in J\setminus\Phi} p_j \sum_{k\in J} \alpha_k + \sum_{j\in J\setminus\Phi} w_j \qquad (6.3.3) \\ &= (s - A)\sum_{j\in J} \alpha_j + 2q + P\sum_{j\in J} \alpha_j - \sum_{j\in J\setminus\Phi} (p_j \sum_{k\in J} \alpha_k - w_j). \end{aligned}$$

Since $t = \sum_{j\in\Phi} p_j + s > A$, the set of the jobs must satisfy $\sum_{j\in J\setminus\Phi} p_j < P - A + s$. Therefore, looking for such a set Φ is equivalent to solving the following knapsack problem, KP2(x):

$$\text{Maximize} \sum_{j\in J\setminus\Phi} (p_j \sum_{k\in J} \alpha_k - w_j), \text{ such that: } \sum_{j\in J\setminus\Phi} p_j < P + s - x, \Phi \subset J, \quad (6.3.4)$$

where $x = A$. In this situation, we set $D = \sum_{j\in\Phi} p_j + s$ using the answer Φ.

If all jobs are delivered in a single batch, then we may have three resulting schedules: (1) setting $D = A$ so that all jobs are early (if $A \geq P + s$), (2) setting $D = A$ so that all jobs are tardy (if $A < P + s$) and (3) setting $D = P + s$ so that all jobs are early (if $A < P + s$). Now we are ready to present our pseudo-polynomial Algorithm CH6-A2.

Algorithm CH6-A2

1. Set $v_1, v_2, v_3, v_I, v_{II} = \infty$ */*Initialization.*

2. If $A \geq P + s$, then set $D = A$ and $v_1 = q$ */*Schedule all jobs early in one batch.*

3. If $A < P + s$, then set $D = A$ and $v_2 = q + \sum_{j=1}^{n} w_j$ */*Schedule all jobs tardy in one batch.*

4. If $A < P + s$, then set $D = P + s$ and $v_3 = q + (D - A)\sum_{j=1}^{n} \alpha_j$ */*Schedule all jobs early in one batch with a revised due date.*

5. If $A < P + s$, then set $x = A$ and solve the KP1(x) problem described in equation (6.3.2). Using the answer Ω, set $D = A$ and calculate the cost of the corresponding schedule $v_I = 2q + \sum_{j\in J\setminus\Omega} w_j$ */*Pattern I.*

6. If $A < P + s$, then set $x = A$ and solve the KP2(x) problem described in equation (6.3.4). Using the answer Φ, set $D = \sum_{j\in\Phi} p_j + s$ and calculate the cost of the corresponding schedule $v_{II} = (D - A)\sum_{j=1}^{n} \alpha_j + \sum_{j\in J\setminus\Phi} w_j + 2q$ */*Pattern II.*

7. Find the optimal solution by $v^* = \min\{v_1, v_2, v_3, v_I, v_{II}\}$ and obtain the corresponding schedule and the corresponding D.

Theorem 6.3.2 *Algorithm CH6-A2 solves the* $1|s, A > 0, CON|\sum \alpha_j R_j + \sum w_j U_j + bq$ *problem in* $O(nP)$ *time, where* $P = \sum_{j=1}^{n} p_j$. *This shows that the problem is* $\mathcal{NP}$*-hard only in the ordinary sense.*

Proof. Many known algorithms for the KP1(x) and KP2(x) problem run in $O(nP)$ time [Pinedo, 2001]. ■

Fully Polynomial Time Approximation Scheme

In order to obtain an FPTAS for the $1|s, A > 0,\text{CON}|\sum \alpha_j R_j + \sum w_j U_j + bq$ problem, we need to solve the minimization version of the KP1(x) and KP2(x) problems approximately. The minimization version of the KP1(x) problem, denoted by MinKP1(x), is shown below:

$$\text{Minimize} \sum_{j \in \bar{\Omega}} w_j, \text{ such that: } \sum_{j \in \bar{\Omega}} p_j \geq P + s - x, \bar{\Omega} \subset J, \tag{6.3.5}$$

where $x = A$. The minimization version of the KP2(x) problem, denoted by MinKP2(x), is shown below:

$$\text{Minimize} \sum_{j \in J \setminus \bar{\Phi}} (p_j \sum_{k \in J} \alpha_k - w_j), \text{ such that: } \sum_{j \in J \setminus \bar{\Phi}} p_j \geq x - s, \bar{\Phi} \subset J, \tag{6.3.6}$$

where $x = A$. We know that the above two problems, MinKP1(x) and MinKP2(x), can be solved with $(1 + \varepsilon)$-approximate ratio by the FPTAS of [Gens and Levner, 1979a] in $O(n^2/\varepsilon)$ time. Now we are ready to present Algorithm CH6-A3, which is subsequently proven to be an FPTAS.

Algorithm CH6-A3

1. Set $v_1, v_2, v_3, v_I, v_{II} = \infty$ /**Initialization.*

2. Do the same as in the steps $2 - 4$ of Algorithm CH6-A2 /**Set* v_1, v_2, v_3.

3. If $A < P + s$, then set $x = A$ and solve the MinKP1(x) problem described in equation (6.3.5) by the FPTAS in [Gens and Levner, 1979a]. Based on the answer $\bar{\Omega}$, set $D = A$ and calculate the cost of the corresponding schedule by $v_I = 2q + \sum_{j\in\bar{\Omega}} w_j$ /*Pattern I.*

4. If $A < P + s$, then set $x = A$ and solve the MinKP2(x) problem described in equation (6.3.6) by the FPTAS in [Gens and Levner, 1979a]. Based on the answer $\bar{\Phi}$, set $D = \sum_{j\in\bar{\Phi}} p_j + s$ and calculate the cost of the corresponding schedule $v_{II} = (D - A)\sum_{j=1}^{n} \alpha_j + \sum_{j\in J\setminus\bar{\Phi}} w_j + 2q$ /*Pattern II.*

5. Find the approximate result by $v_{apx} = \min\{v_1, v_2, v_3, v_I, v_{II}\}$ and obtain the corresponding schedule and the corresponding D.

Theorem 6.3.3 *Algorithm CH6-A3 provides a $(1+\varepsilon)$-approximate solution for the $1|s, A > 0, CON|\sum \alpha_j R_j + \sum w_j U_j + bq$ problem in $O(n^2/\varepsilon)$ time. This shows that the algorithm is an FPTAS.*

Proof. Since step 2 runs in $O(n)$ time and steps 3 and 4 run in $O(n^2/\varepsilon)$ time [Gens and Levner, 1979a], the overall running time is $O(n^2/\varepsilon)$. ■

6.4 Time-constrained Problem

In the time-constrained problem, $1|s, T, A, \text{CON}|\sum \alpha_j R_j + \sum w_j U_j + bq$, if there are two batches, then the time elapsed between two deliveries must be at least T. If $T = 1$, it is equivalent to the unconstrained problem. We assume, therefore, that $T > 1$. Notice that Proposition 6.3.1 for the unconstrained problem is applicable for this problem as well, since it makes no sense to deliver early or tardy jobs in two different batches. The following lemma allows us to implement slightly different versions of Algorithm CH6-A1, Algorithm CH6-A2 and Algorithm CH6-A3 to solve the time-constrained problem.

Lemma 6.4.1 *For any given schedule that is feasible for the unconstrained problem and has one or two batches, we can convert it into a schedule that is feasible for the time-constrained with the scheduling cost unchanged.*

Proof. If the schedule has only one batch, it is clearly true that it is also feasible for the time-constrained problem. If the schedule has two batches and the first delivery happens at time t, then let us consider two cases. Notice that in the unconstrained problem we always assume that there is no idle time in any schedule, and thus the second delivery always happens at time $P + 2s$. If $P + 2s - t \geq T$, then the schedule automatically satisfies the time constraint. If $P + 2s - t < T$, then we just need to add an idle time period $T - (P + 2s - t)$ between the first and the second delivery. This makes a time period T between these two deliveries and thus makes the schedule feasible for the time-constrained problem. Since the second batch is a tardy batch, then this delay will not add any extra cost for the schedule. ■

Based on the above proof, for any schedule found by Algorithm CH6-A1, Algorithm CH6-A2 and Algorithm CH6-A3 for the unconstrained problem, if there are two batches and the first delivery happens at t with $P + 2s - t < T$, then let the algorithms further add an idle time period $T - (P + 2s - t)$ just before the second delivery. Therefore, the resulting schedule is feasible for the time-constrained problem as well. Let us name the corresponding algorithms with this operation as Algorithm CH6-A4, Algorithm CH6-A5 and Algorithm CH6-A6. The following three corollaries directly follow from the proofs of the corresponding algorithms and the above discussion.

Corollary 6.4.1 *Algorithm CH6-A4 solves the* $1|s, T, A = 0, CON|\sum \alpha_j R_j + \sum w_j U_j +$ *bq problem in polynomial time.*

Corollary 6.4.2 *Algorithm CH6-A5 solves the* $1|s, T, A > 0, CON|\sum \alpha_j R_j + \sum w_j U_j +$ *bq problem in* $O(nP)$ *time, where* $P = \sum_{j=1}^{n} p_j$. *This shows that the problem is* $\mathcal{NP}$*-hard only in the ordinary sense.*

Corollary 6.4.3 *Algorithm CH6-A6 provides a $(1+\varepsilon)$-approximate solution for the $1|s,T,A>0,CON|\sum \alpha_j R_j + \sum w_j U_j + bq$ problem in $O(n^2/\varepsilon)$ time. This shows that the algorithm is an FPTAS.*

6.5 Capacity-constrained Problem

Now we study the capacity-constrained problem, $1|s,B,A,\text{CON}|\sum \alpha_j R_j + \sum w_j U_j + bq$. If $B = n$, it is equivalent to the unconstrained problem. We assume, therefore, that $B < n$.

6.5.1 Pseudo-polynomial Algorithm

Let us first observe the following easy-to-prove structure for optimal schedules.

Proposition 6.5.1 *For the $1|s,B,A,CON|\sum \alpha_j R_j + \sum w_j U_j + bq$ problem, there is an optimal schedule where all tardy jobs (if any) are scheduled following all early jobs.*

Proof. Since all jobs have a common assigned due date, tardy jobs (if any) must be scheduled at the end. ■

Since all jobs share a common due date, ordering does not matter for either early jobs or tardy jobs in any optimal schedule. Without loss of generality, we assume that all jobs are indexed as: $p_1 \leq p_2 \leq ... \leq p_n$. Let (j,k_1,k_2,t,v), $t \geq 0$, represent a partial schedule on job set $\{1,...,j\}$ with cost v and makespan t, in which k_1 is the number of early jobs in the current early batch and k_2 is the number of tardy jobs in the current tardy batch. Here v includes the penalties for tardy jobs and the batch-delivery costs of early jobs or tardy jobs from $\{1,...,j\}$. For a partial schedule (j,k_1,k_2,t,v), we can generate another partial schedule by scheduling job $j+1$ in the following five ways: (1) as an early job in the current early batch, or (2) as an early job starting a new early batch, or (3) as a tardy job in the current tardy batch, or (4) as a tardy job starting a new tardy batch, or (5) completing (j,k_1,k_2,t,v)

into a full schedule $(n, 0, 0, -1, v + v_{(j,k_2)})$ by setting $D = t$, delivering the last early batch at t and scheduling jobs $\{j+1, ..., n\}$ as tardy jobs after t using the minimum number of tardy batches, where we use the first four entries $\{n, 0, 0, -1\}$ to specify a full schedule. In this last case,

$$v_{(j,k_2)} = \max\{t - A, 0\} \sum_{j=1}^{n} \alpha_j + q\lceil \frac{n-j+k_2}{B} \rceil + \sum_{i=j+1}^{n} w_i,$$

where $\max\{t-A, 0\} \sum_{j=1}^{n} \alpha_j$ is the due-date-assignment cost; $q\lceil \frac{n-j+k_2}{B} \rceil$ is the batch-delivery cost for all tardy jobs and $\sum_{i=j+1}^{n} w_i$ is the tardy penalty of jobs $\{j+1, ..., n\}$. In the following Algorithm CH6-A7, we continuously update the state (n, v) by the smallest total cost v for any full schedule.

Consider two states (j, k_1, k_2, t, v) and (j, k_1, k_2, t, v') with $v \leq v'$. Then we can eliminate the second state, because any later states generated from the second state can not lead to a smaller v value than the value of similar states generated from the first state.

Remark 6.5.1 *For any two states with the same entries $(j, k_1, k_2, t, \cdot)$, we can eliminate the one with larger v.*

Algorithm CH6-A7 starts from an empty schedule $(0, 0, 0, s, q)$, where s and q represent the batch-setup time and batch-delivery cost for the first early batch, respectively. To begin, we set $v^* = \sum_{j=1}^{n} w_j + q\lceil \frac{n}{B} \rceil$ as the initial value of optimal solution, which is the minimum cost of schedules in which all jobs are tardy. During the algorithm, v^* will be updated when we complete a partial schedule into a full schedule with the total cost smaller than v^*. In Algorithm CH6-A7, we start a new early or tardy batch only when every previously scheduled early or tardy batch has exactly B jobs.

Algorithm CH6-A7

[Initialization]: Set $\mathcal{S}^{(0)} = \{(0, 0, 0, s, q)\}$, $v^* = \sum_{j=1}^{n} w_j + q\lceil \frac{n}{B} \rceil$.

[Generalization]: Generate set $\mathcal{S}^{(j)}$ from $\mathcal{S}^{(j-1)}$.

For $j = 1$ **to** $n+1$

Set $\mathcal{T} = \emptyset$.

For each state $(j-1, k_1, k_2, t, v)$ in $\mathcal{S}^{(j-1)}$

1. If $k_1 < B$ and $j \leq n$, then set $\mathcal{T} \leftarrow \mathcal{T} \cup (j, k_1+1, k_2, t+p_j, v)$ /*Job j is scheduled in the current early batch.*
2. If $k_1 = B$ and $j \leq n$, then set $\mathcal{T} \leftarrow \mathcal{T} \cup (j, 1, k_2, t+p_j+s, v+q)$ /*Job j is scheduled in a new early batch.*
3. If $k_2 < B$ and $j \leq n$, then set $\mathcal{T} \leftarrow \mathcal{T} \cup (j, k_1, k_2+1, t, v+w_j)$ /*Job j is scheduled in the current tardy batch.*
4. If $k_2 = B$ and $j \leq n$, then set $\mathcal{T} \leftarrow \mathcal{T} \cup (j, k_1, 1, t, v+q+w_j)$ /*Job j is scheduled in a new tardy batch.*
5. Set $D = t$ and update $v^* = \min\{v^*, v_{(j-1,k_1,k_2,t)} + v_{(j,k_2)}\}$ /*Complete the partial schedule into a full schedule.*

Endfor

[Elimination]: If $j \leq n$, for any two states (j, k_1, k_2, t, v) and (j, k_1, k_2, t, v') with $v \leq v'$, eliminate the one with v' from $\mathcal{T}$ (Remark 6.5.1), and set $\mathcal{S}^{(j)} = \mathcal{T}$.

Endfor

[Optimization]: Trace back v^* to obtain D and the corresponding optimal schedule.

Theorem 6.5.1 *For the* $1|s, B, A, CON|\sum \alpha_j R_j + \sum w_j U_j + bq$ *problem, Algorithm CH6-A7 finds an optimal schedule in* $O(n[P+ns])$ *time, where* $P = \sum_{j=1}^{n} p_j$. *This shows that the problem is* $\mathcal{NP}$*-hard only in the ordinary sense.*

Proof. For each state (j, k_1, k_2, t, v) in $\mathcal{S}^{(j)}$, there are at most five operations. k_1, k_2 take at most B and j takes at most $n+1$ as the largest value. By the [Elimination] step, there are at most $[P+ns]$ different $\{j, k_1, k_2, t, v\}$ with the same j, k_1 and k_2, but different t values. Thus the running time is $O(nB^2[P+ns]) = O(n[P+ns])$, as B is fixed. ■

6.5.2 Fully Polynomial Time Approximation Scheme

In this section, we first provide an approximation algorithm for the problem of minimizing the total cost of the weighted number of tardy jobs and batch-delivery costs, where all jobs have a *fixed* common due date x and the maximum number of jobs in a single batch is B, denoted by $1|s, B, d_j = x|\sum w_j U_j + bq$. Then we provide an FPTAS for the capacity-constrained problem using this algorithm for the $1|s, B, d_j = x|\sum w_j U_j + bq$ problem.

FPTAS for the $1|s, B, d_j = x|\sum w_j U_j + bq$ Problem

Theorem 6.5.2 *The $1|s, B, d_j = x|\sum w_j U_j + bq$ problem is $\mathcal{NP}$-hard.*

Proof. Consider the knapsack problem shown below, which is well-known to be $\mathcal{NP}$-hard:

$$\text{Maximize } \sum_{j\in\Omega} w_j, \text{ such that: } \sum_{j\in\Omega} p_j \leq x, \Omega \subseteq J. \tag{6.5.7}$$

Consider an instance of the $1|s, B, d_j = x|\sum w_j U_j + bq$ problem, in which $s = q = 0$ and $B = n$. Then any optimal schedule for the above instance has the total processing time of early jobs less than or equal to x. Define the job set formed by all early jobs in any optimal schedule as Ω. Then Ω is, indeed, an optimal solution to the knapsack problem. We have proven that the knapsack problem reduces the $1|s, B, d_j = x|\sum w_j U_j + b$ problem. ■

We next find an approximation solution to the $1|s, B, d_j = x|\sum w_j U_j + bq$ problem in three steps: (1) we determine initial bounds such that $v' \leq u(x) \leq nv'$, where $u(x)$ is the optimal solution value to the $1|s, B, d_j = x|\sum w_j U_j + bq$ problem; (2) we narrow the bounds into tight bounds such that $\bar{v} \leq u(x) \leq 3\bar{v}$; (3) we approximate the optimal solution using $[\bar{v}, 3\bar{v}]$.

Step I: Initial Bounds Initial bounds can be obtained by solving an auxiliary problem, where we need to minimize the maximum weight of tardy jobs, denoted by $1|s, B, d_j = x|\max w_j U_j$. Let $w(x)$ be the optimal solution value. Then jobs in

$J_{(w(x))} = \{j|w_j > w(x), j \in J\}$ have to be delivered before or at x. This requires $x \geq \sum_{j \in J_{(w(x))}} p_j + \lceil \frac{|J_{(w(x))}|}{B} \rceil s$. In order to determine $w(x)$, Algorithm CH6-A8(x) performs binary search on $\{w_1, w_2, ..., w_n\}$, where $w_1 \leq w_2 \leq ... \leq w_n$.

Algorithm CH6-A8(x)

[Initialization] Set $L_1 = 0$, $L_2 = n$ and $k(x) = 0$.

While $L_1 < L_2$, **do**

1. Set $k = \lceil (k_1 + k_2)/2 \rceil$ and $J_{(w_k)} = \{j|w_j > w_k, j \in J\}$;
2. If $x < \sum_{j \in J_{(w_k)}} p_j + \lceil \frac{|J_{(w_k)}|}{B} \rceil s$, then set $L_1 = k$;
3. If $x \geq \sum_{j \in J_{(w_k)}} p_j + \lceil \frac{|J_{(w_k)}|}{B} \rceil s$, then set $k(x) = k$ and $L_2 = k$.

Endwhile

[Result] If $k(x) = 0$, then set $w(x) = 0$; Otherwise, set $w(x) = w_{k(x)}$.

Lemma 6.5.1 *Algorithm CH6-A8(x) finds $w(x)$, the optimal solution value for the $1|s, B, d_j = x|\max w_j U_j$ problem in $O(n \log n)$ time. Furthermore, $w(x)$ generates initial bounds for $u(x)$ such that $v' \leq u(x) \leq nv'$, where $v' = w(x) + q$. (Recall that $u(x)$ is the optimal solution to the $1|s, B, d_j = x|\sum w_j U_j + bq$ problem.)*

Proof. All calculations in Algorithm CH6-A8(x) can be done in $O(n)$ time per iteration and the binary search runs in $O(\log n)$ time. Then the overall running time is $O(n \log n)$. Since there is at least one batch in any schedule for the $1|s, B, d_j = x|\sum w_j U_j + bq$ problem, the lower bound is $v' = w(x) + q$. Since there are at most n batches in any schedule for the $1|s, B, d_j = x|\sum w_j U_j + bq$ problem, the upper bound is $nw(x) + nq = nv'$. ■

Step II: Tight Bounds To narrow the initial bounds $[v', nv']$, we develop the following Algorithm CH6-A9(x, u, ε) that, for given $u > 0$ and small $\varepsilon > 0$, proves either $u(x) > (1 - \varepsilon)u$ or $u(x) \leq u$. Recall that $u(x)$ is the optimal solution to the $1|s, B, d_j = x|\sum w_j U_j + bq$ problem. Algorithm CH6-A9(x, u, ε) uses the same

state representation (j, k_1, k_2, t, v) as in Algorithm CH6-A7 and starts from an empty schedule $(0, 0, 0, s, q)$, where s and q are for the first early batch. Since the algorithm has already assumed at least one early batch, let us define $\bar{v}(x) = \sum_{j=1}^{n} w_j + q\lceil \frac{n}{B} \rceil$ the cost of a full schedule with all tardy jobs.

Algorithm CH6-A9(x, u, ε)

[Initialization]: Set $\mathcal{S}^{(0)} = \{(0, 0, 0, s, q)\}$, $\bar{v}(x) = \sum_{j=1}^{n} w_j + q\lceil \frac{n}{B} \rceil$ and $\mathcal{S}^{(j)} = \emptyset$, $j = 1, ..., n$.

[Partitioning]: Partition $[0, u]$ into $\lceil n/\varepsilon \rceil$ equal subintervals of size $\varepsilon u/n$ with the last one possibly smaller.

[Generation]: Generate set $\mathcal{S}^{(j)}$ from $\mathcal{S}^{(j-1)}$.

For $j = 1$ **to** n

Set $\mathcal{T} = \emptyset$.

For each state $(j-1, k_1, k_2, t, v)$ in $\mathcal{S}^{(j-1)}$

1. If $k_1 < B$ and $t + p_j \leq x$, then set $\mathcal{T} \leftarrow \mathcal{T} \cup (j, k_1 + 1, k_2, t + p_j, v)$ */*Schedule job j early in the current early batch.*
2. If $k_1 = B$ and $t + p_j + s \leq x$, then set $\mathcal{T} \leftarrow \mathcal{T} \cup (j, 1, k_2, t + p_j + s, v + q)$ */*Schedule job j early starting a new early batch.*
3. If $k_2 < B$, then set $\mathcal{T} \leftarrow \mathcal{T} \cup (j, k_1, k_2 + 1, t, v + w_j)$ */*Schedule job j as a tardy job in the current tardy batch.*
4. If $k_2 = B$, then set $\mathcal{T} \leftarrow \mathcal{T} \cup (j, k_1, 1, t, v + q + w_j)$ */*Schedule job j as a tardy job starting a new tardy batch.*

Endfor

[Elimination]: Update set $\mathcal{S}^{(j)}$.

1. Delete states (j, k_1, k_2, t, v) with $v > u$.

2. If more than one state has a v value that falls into the same subinterval of $[0, u]$, then discard all but one of these states, keeping only the representative state with the smallest t coordinate for each interval.
3. For any two states (j, k_1, k_2, t, v) and (j, k_1, k_2, t, v') with $v \leq v'$, eliminate the one with v' from set $\mathcal{T}$. (Remark 6.5.1)
4. Set $\mathcal{S}^{(j)} = \mathcal{T}$.

Endfor

[Result]: If $\mathcal{S}^{(n)} = \emptyset$ and $\bar{v}(x) > u$, then deduce $u(x) > (1-\varepsilon)u$; If $\mathcal{S}^{(n)} \neq \emptyset$ or $\bar{v}(x) \leq u$, then deduce $u(x) \leq u$.

Theorem 6.5.3 *In $O(n^2/\varepsilon)$ time, Algorithm CH6-A9(x, u, ε) reports: either $u(x) > (1-\varepsilon)u$, if $\mathcal{S}^{(n)} = \emptyset$ and $\bar{v}(x) > u$; or $u(x) \leq u$, if $\mathcal{S}^{(n)} \neq \emptyset$ or $\bar{v}(x) \leq u$. (Recall that $u(x)$ is the optimal solution to $1|s, B, d_j = x|\sum w_j U_j + bq$ and $\bar{v}(x) = \sum w_j + q\lceil \frac{n}{B} \rceil$.)*

Proof. Similarly to Theorem 4.4.3, if $\mathcal{S}^{(n)} \neq \emptyset$, then there is at least one state (n, k_1, k_2, t, v) that has not been eliminated. Therefore, $u(x) \leq v \leq u$. Since $\bar{v}(x)$ represents the value of a feasible schedule with all tardy jobs for the problem, we know $u(x) \leq \bar{v}(x)$. Then $\bar{v}(x) \leq u$ implies $u(x) \leq u$. If $\mathcal{S}^{(n)} = \emptyset$, then, by [Partitioning] and [Elimination], the error introduced is at most εu. Therefore, $v > (1-\varepsilon)u$. Since $u(x) = \min\{v, \bar{v}(x)\}$ and $\bar{v}(x) > u$, then $u(x) > (1-\varepsilon)u$. For each state (j, k_1, k_2, t, v) in $\mathcal{S}^{(j)}$, there are at most five operations. k_1, k_2 take at most B values and j takes at most n values. By the [Partitioning] and [Elimination] steps, there are at most $\lceil n/\varepsilon \rceil$ different $\{j, k_1, k_2\}$ triples. Thus the time complexity is $O(n^2/\varepsilon)$. ■

Again, we consider to use $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ introduced in Chapter 3 to narrow the bounds $[v', nv']$. Let $a_1 = v'$, $a_2 = nv'$, $\omega = u$ and $\mathcal{P}$ be the $1|s, B, d_j = x|\sum w_j U_j + bq$ problem. Then Algorithm CH6-A9$(x, u, 1/3)$ can be used as $\mathcal{A}(\omega, \mathcal{P})$. Since Algorithm CH6-A9$(x, u, 1/3)$ runs in $O(n^2)$ time, so in $O(n^2 \log\log n)$ time, $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ reports ξ that implies a pair of tight bounds $[\xi, 3\xi]$. In this situation, let us refer to $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ as Algorithm CH6-A10(x).

Corollary 6.5.1 *Algorithm CH6-A10(x) can narrow the bounds* $[v', nv']$ *into* $[\bar{v}, 3\bar{v}]$ *in* $O(n^2 \log\log n)$ *time by setting* $\bar{v} = \xi$.

Step III: Fully Polynomial Time Approximation Scheme For any given $\varepsilon > 0$, using the bounds $\bar{v} \leq u(x) \leq 3\bar{v}$ obtained by Algorithm CH6-A10(x), we run a slightly changed version of Algorithm CH6-A9($x, u, 1/3$), called Algorithm CH6-A11(x), with $u = (1 + \varepsilon/3)3\bar{v}$: The only difference is that in the [Partitioning] step we partition $[0, u] = [0, (1+\varepsilon/3)3\bar{v}]$ into $n\lceil 3/\varepsilon + 1\rceil$ intervals of size at most $\varepsilon\bar{v}/n$, so the cumulative error over n iterations will be no more than $\varepsilon\bar{v}$. Since we know that the problem has an optimal solution value $u(x) \leq 3\bar{v}$, the algorithm will find a solution $v(x)$ for the $1|s, B, d_j = x|\sum w_j U_j + bq$ problem such that $v(x) \leq 3\bar{v} + \varepsilon\bar{v} = u$, which means that the algorithm will never end with an empty $\mathcal{T}^*$. Furthermore, whichever subinterval of $[\bar{v}, 3\bar{v}]$ $u(x)$ falls into, the algorithm will generate an approximate solution $v(x)$ with at most $\varepsilon\bar{v}$ error away from it, i.e., $v(x) \leq u(x) + \varepsilon\bar{v} \leq (1 + \varepsilon)u(x)$. Therefore, we have the following corollary.

Corollary 6.5.2 *For any given* $\varepsilon > 0$, *Algorithm CH6-A11(x) finds a* $(1 + \varepsilon)$*-approximate solution for the* $1|s, B, d_j = x|\sum w_j U_j + bq$ *problem in* $O(n^2/\varepsilon)$ *time.*

Now we are ready to present an FPTAS for the problem.

Algorithm CH6-A12(x)

[InitialBounds]: Run Algorithm CH6-A8(x) and set $v' = w(x) + q$;

[TightBounds]: Run Algorithm CH6-A10(x) and obtain: $\bar{v} \leq u(x) \leq 3\bar{v}$.

[Approximation]: Run Algorithm CH6-A11(x) and obtain an approximate schedule.

Theorem 6.5.4 *Algorithm CH6-A12(x) is an FPTAS, which finds an* $(1+\varepsilon)$*-approximate solution to the* $1|s, B, d_j = x|\sum w_j U_j + bq$ *problem in* $O(n^2 \log\log n + n^2/\varepsilon)$ *time.*

Proof. The complexity and correctness follow from the component algorithms. ■

FPTAS for the Capacity-constrained Problem

In order to find an approximate solution for the capacity-constrained problem, we are going to determine a way to enumerate a number of particular values of x such that both the total number of enumerated x and the total error introduced into the final solution value are controllable. Then we can implement Algorithm CH6-A12(x) to approximate each constructed $1|s, B, d_j = x|\sum w_j U_j + bq$ problem.

For a given $\varepsilon > 0$, let us divide the interval $(A, P + \lceil \frac{n}{B} \rceil s]$ into l mutually exclusive subintervals as follows:

$$\begin{aligned} H_1^B &= (A, A + (1+\varepsilon)]; \\ H_2^B &= (A + (1+\varepsilon), A + (1+\varepsilon) + (1+\varepsilon)^2]; \\ &\cdots\cdots \\ H_l^B &= (A + \sum_{k=1}^{l-1}(1+\varepsilon)^k, \min\{P + \lceil \frac{n}{B} \rceil s, A + \sum_{k=1}^{l}(1+\varepsilon)^k\}], \end{aligned} \tag{6.5.8}$$

where l is such that

$$A + \sum_{k=1}^{l-1}(1+\varepsilon)^k < P + \lceil \frac{n}{B} \rceil s \leq A + \sum_{k=1}^{l}(1+\varepsilon)^k$$

and thus

$$l = \lceil \log_{1+\varepsilon}[(P + \lceil \frac{n}{B} \rceil s - A + 1)\varepsilon + 1] \rceil - 1.$$

Let $\sigma(t)$ be an optimal schedule for the capacity-constrained problem with $D(\sigma(t)) = t \in H_k^B$ for some $k \in [1, l]$. We have the total cost $(t - A)\sum_{j=1}^{n} \alpha_j + u(\sigma(t))$, where $u(\sigma(t)) = \sum_{j=1}^{n} w_j U_j(\sigma(t)) + b(\sigma(t))q$. Schedule $\sigma(t)$ must also be an optimal schedule for the $1|s, B, d_j = t|\sum w_j U_j + bq$ problem. Let

$$t' = \begin{cases} A + \sum_{i=1}^{k}(1+\varepsilon)^i, & \text{if } 1 \leq k < l \\ \min\{P + \lceil \frac{n}{B} \rceil s, A + \sum_{k=1}^{l}(1+\varepsilon)^k\}, & \text{if } k = l \end{cases} \tag{6.5.9}$$

be the right end point of the interval H_k^B. By equation (6.5.8), we have

$$t' - A \leq (1+\varepsilon)(t - A). \tag{6.5.10}$$

Consider a schedule $\sigma(t')$ with $D(\sigma(t')) = t'$ for the capacity-constrained problem. Suppose $\sigma(t')$ is an optimal schedule for the $1|s, B, d_j = t'|\sum w_j U_j + bq$ problem and the cost is $u(\sigma(t')) = \sum_{j=1}^{n} w_j U_j(\sigma(t')) + b(\sigma(t'))q$. Then because $t' > t$, we have

$$u(\sigma(t')) \leq u(\sigma(t)). \tag{6.5.11}$$

Let $\sigma'(t')$ be an approximation schedule for the $1|s, B, d_j = t'|\sum w_j U_j + bq$ problem found by Algorithm CH6-A12(x) with the cost $u(\sigma'(t'))$. Therefore, we have the following total cost

$$(t' - A)\sum_{j=1}^{n} \alpha_j + u(\sigma'(t')) \leq (1+\varepsilon)(t - A)\sum_{j=1}^{n} \alpha_j + (1+\varepsilon)u(\sigma(t')) \tag{6.5.12}$$

$$\leq (1+\varepsilon)[(t - A)\sum_{j=1}^{n} \alpha_j + u(\sigma(t))], \tag{6.5.13}$$

where (6.5.12) is true by equation (6.5.10) and Theorem 6.5.3, and (6.5.13) holds by equation (6.5.11).

Algorithm CH6-A13 solves the $1|s, B, d_j = t'|\sum w_j U_j + bq$ problems approximately for all t' defined in equation (6.5.9) and $t' = A$, where $t' = A$ leads to a zero due-date-assignment cost. Then the approximate solution to the $1|s, B, d_j = A|\sum w_j U_j + bq$ problem provides an approximate solution to the original capacity-constrained problem. For all the resulting schedules, we set $D = t'$.

Algorithm CH6-A13

1. Set $\mathcal{T} = \emptyset$;
2. For each $x = A$ and $x = t$ defined in equation (6.5.9), construct the $1|s, B, d_j = x|\sum w_j U_j + bq$ problem;
3. Run Algorithm CH6-A12(x) and let $v(x)$ be the smallest v value in $S^{(n)}$. Thus set $\mathcal{T} \leftarrow \mathcal{T} \cup (x, v)$ and $v = \sum_{j=1}^{n} \alpha_j(x - A) + v(x)$.
4. Select the state with the smallest v from $\mathcal{T}$ and obtain the corresponding schedule.

Theorem 6.5.5 *Given $\varepsilon > 0$, Algorithm CH6-A13 provides an $(1+\varepsilon)$-approximate solution to the $1|s, B, A, CON|\sum \alpha_j R_j + \sum w_j U_j + bq$ problem in $O(l[n^2 \log\log n + n^2/\varepsilon])$ time, where $l = \lceil \log_{1+\varepsilon}[(P + \lceil \frac{n}{B} \rceil s - A + 1)\varepsilon + 1] \rceil - 1$.*

Proof. Algorithm CH6-A13 calls Algorithm CH6-A12(x) $l+2$ times and Algorithm CH6-A12(x) runs in $O(n^2 \log\log n + n^2/\varepsilon)$ time. Thus Algorithm CH6-A13 runs in $O(l[n^2 \log\log n + n^2/\varepsilon])$ time. ■

6.6 Summary

In this chapter, we studied three supply chain scheduling problems with CON due date assignment and constraints on deliveries. For the unconstrained problem, the time-constrained problem and the capacity-constrained problem, we first proved that they are $\mathcal{NP}$-hard. Then we proposed pseudo-polynomial algorithms to establish that each problem is $\mathcal{NP}$-hard only in the ordinary sense. Finally, we presented fully polynomial time approximation schemes for them.

Chapter 7

Models with Distinct Assignable Due Dates

In Chapter 6, we have studied four supply chain scheduling problems with delivery costs and a common assignable due date (the CON problems). In this chapter, we change our focus to the DIF problems where arbitrary due dates are allowed to be assigned to jobs individually.

7.1 Introduction

Shabtay and Steiner [2006] study 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 combination of the problem in Chapter 3 and the problem studied in the paper by Shabtay and Steiner [2006].

This chapter is organized as follows. Section 7.2 contains preliminaries: problem definition, some important propositions about decisions on due dates and a strong $\mathcal{NP}$-hardness proof for our problem. In Section 7.3, we study the case with a uniform due-date-assignment cost. We first prove that it is $\mathcal{NP}$-hard, and then we show that it is $\mathcal{NP}$-hard only in the ordinary sense by presenting a pseudo-polynomial algorithm, which requires only polynomial time when all processing times are equal. Finally, we convert the pseudo-polynomial algorithm into an FPTAS. In Section 7.4, we present a polynomial algorithm for the case with equal due-date-assignment costs and equal tardiness penalties. Section 7.5 includes our final conclusions.

7.2 Preliminaries

Note that all the terminologies, assumptions and notations introduced in Chapters 1 and 3 are applied in this chapter. Our goal is to find a schedule which minimizes the sum of the due-date-assignment costs, the weighted number of tardy jobs and the batch-delivery costs, denoted by $1|s, A, \text{DIF}|\sum \alpha_j R_j + \sum w_j U_j + bq$. There are three levels of decisions to be made in the $1|s, A, \text{DIF}|\sum \alpha_j R_j + \sum w_j U_j + bq$ problem:

(1) determining a job sequence;

(2) grouping the sequence into batches;

(3) assigning arbitrary due dates to each job individually.

Suppose that we are given a schedule σ, where the first two (1 and 2) 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 $b(\sigma)$ and the batch-delivery costs $b(\sigma)q$. In order to minimize the total cost, we need to determine $D_j(\sigma)$ for each j, which minimizes the cost $\sum \alpha_j R_j(\sigma) + \sum w_j U_j(\sigma)$.

In Figure 7.1 (a), we use the bold and sloped line to represent the cost, which

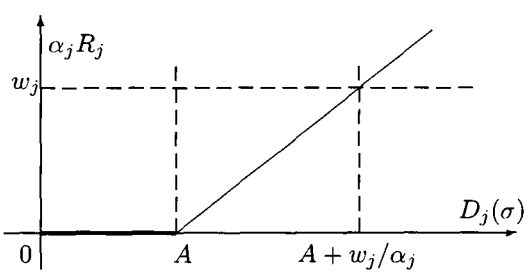
is a function of $D_j(\sigma)$:

$$\alpha_j R_j(D_j(\sigma)) = \begin{cases} 0, & \text{if } D_j(\sigma) \leq A \text{ (the bold line)} \\ \alpha_j(D_j(\sigma) - A), & \text{if } D_j(\sigma) > A \text{ (the sloped line)} \end{cases} . \quad (7.2.1)$$

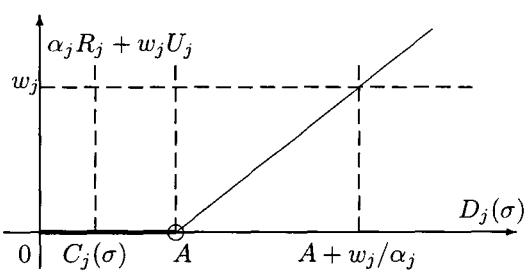
Based on equation (7.2.1) , we use the bold and sloped lines in Figure 7.1 (b), (c) and (d) to represent the total cost for job j as a function of $D_j(\sigma)$:

$$\alpha_j R_j(D_j(\sigma)) + w_j U_j(D_j(\sigma)) = \begin{cases} \alpha_j \max\{D_j(\sigma) - A, 0\} + w_j, & \text{if } D_j(\sigma) < C_j(\sigma) \\ \alpha_j \max\{D_j(\sigma) - A, 0\}, & \text{if } D_j(\sigma) \geq C_j(\sigma) \end{cases} , \quad (7.2.2)$$

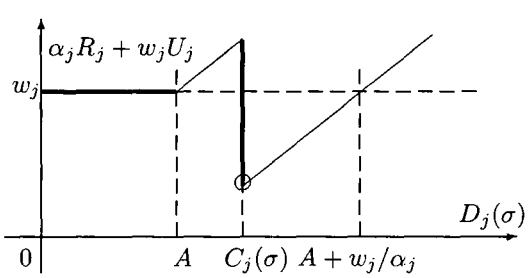
where $C_j(\sigma)$ is the batch-completion time of job j in schedule σ.



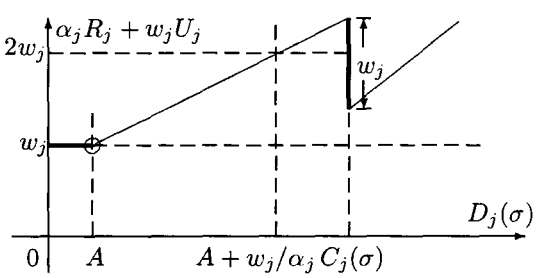
(a) $\alpha_j R_j$ as a function of $D_j(\sigma)$



(b) The cost when $C_j(\sigma) \leq A$



(c) The cost when $A < C_j(\sigma) \leq A + w_j/\alpha_j$



(d) The cost when $C_j(\sigma) > A + w_j/\alpha_j$

Figure 7.1: Due date assignment for job j based on a given schedule σ.

In each figure, the circle represents the point where the minimum cost value and the corresponding assigned due date occur.

When $C_j(\sigma) \leq A$ (Figure 7.1 (b)), if $D_j(\sigma) \leq A$, then job j is early and the cost is 0; if $D_j(\sigma) > A$, then job j is still early, but the cost is $\alpha_j(D_j(\sigma) - A)$. So the minimum cost of 0 occurs when $D_j(\sigma) = A$ (represented by the circle). If $A \leq C_j(\sigma) \leq A + \frac{w_j}{\alpha_j}$ (Figure 7.1 (c)), if $D_j(\sigma) \leq A$, then job j is tardy and the cost is w_j; if $A < D_j(\sigma) < C_j(\sigma)$, then job j is still tardy but the cost is $\alpha_j(D_j(\sigma) - A) + w_j$; if $D_j(\sigma) \geq C_j(\sigma)$, then job j is early and the cost is $\alpha_j(D_j(\sigma) - A)$. So the minimum cost of $\alpha_j(C_j(\sigma) - A)$ occurs when $D_j(\sigma) = C_j(\sigma)$ (represented by the circle) and job j is early. If $C_j(\sigma) > A + \frac{w_j}{\alpha_j}$ (Figure 7.1 (d)), if $D_j(\sigma) \leq A$, then job j is tardy and the cost is w_j; if $A < D_j(\sigma) < C_j(\sigma)$, then job j is still tardy but the cost is $\alpha_j(D_j(\sigma) - A) + w_j$; if $D_j(\sigma) \geq C_j(\sigma)$, then job j is early and the cost is $\alpha_j(D_j(\sigma) - A)$. So the minimum cost of w_j happens when $D_j(\sigma) = A$ (represented by the circle) and job j is tardy. In summary, we have the following proposition first developed by Shabtay and Steiner [2006].

Proposition 7.2.1 *For any given schedule σ, the optimal due date assignment is*

$$D_j(\sigma) = \begin{cases} A, & \text{if } C_j(\sigma) \leq A \text{ or } C_j(\sigma) > A + \frac{w_j}{\alpha_j} \\ C_j(\sigma), & \text{if } A < C_j(\sigma) \leq A + \frac{w_j}{\alpha_j} \end{cases}. \tag{7.2.3}$$

The above Proposition 7.2.1 also implies the following useful observation.

Proposition 7.2.2 *For the $1|s, A, DIF|\sum \alpha_j R_j + \sum w_j U_j + bq$ problem, there is an optimal schedule in which $A + \frac{w_j}{\alpha_j}$ 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 $C_j(\sigma) > A + \frac{w_j}{\alpha_j}$.*

Hall and Potts [2003] proved that the supply chain scheduling problem of minimizing the weighted sum of total completion times and batch-delivery costs, denoted by $1|s'|\sum w'_j C'_j + b'q'$, is strongly $\mathcal{NP}$-hard even with $s' = 0$, where w'_j is the weight, C'_j is the *batch-completion time* of job j, q' is the *batch-delivery cost* and b' is the total number of batches. In the following theorem we prove that the $1|s, A, \text{DIF}|\sum \alpha_j R_j + \sum w_j U_j + bq$ problem is also strongly $\mathcal{NP}$-hard by reducing the

$1|s'=0|\sum w'_jC'_j+b'q'$ problem to an instance of the $1|s,A,\text{DIF}|\sum \alpha_jR_j+\sum w_jU_j+bq$ problem.

Theorem 7.2.1 *The* $1|s,A,DIF|\sum \alpha_jR_j+\sum w_jU_j+bq$ *problem is strongly* $\mathcal{NP}$*-hard.*

Proof. For an arbitrary instance of the $1|s'=0|\sum w'_jC'_j+b'q'$ problem, we construct an instance of the $1|s,A,\text{DIF}|\sum \alpha_jR_j+\sum w_jU_j+bq$ problem by letting $A=0$, $s=s'=0$, $q=q'$,$b=b'$, $\alpha_j=w'_j$ and $w_j=M\alpha_j$, $\forall j\in J$, where $M>P=\sum_{j=1}^{n}p_j$ and p_j is the processing time of job j in both problem instances.

Consider the above instance of the $1|s,A,\text{DIF}|\sum \alpha_jR_j+\sum w_jU_j+bq$ problem. Since $s=0$, all schedules have the fixed makespan P. Then using equation (7.2.3) with $A=0$, the due-date-assignment cost for a job j in any schedule σ is at most $\alpha_jC_j(\sigma)\leq\alpha_jP<\alpha_jM=w_j$. This implies that no job would be tardy in any optimal schedule. Since $A=0$, equation (7.2.3) would yield $D_j(\sigma)=C_j(\sigma)$ $\forall j\in J$ in any optimal schedule. Therefore, we have the following equivalence for any schedule σ:

$$\sum w'_jC'_j(\sigma)+b'q'=\sum \alpha_jC_j(\sigma)+bq=\sum \alpha_jR_j(\sigma)+\sum w_jU_j(\sigma)+bq.$$

Thus the two instances have the same set of feasible schedules and objective value on these. It is easy to see that the $1|s,A,\text{DIF}|\sum \alpha_jR_j+\sum w_jU_j+bq$ problem is in class $\mathcal{NP}$ and the above reduction can be done in $O(n)$ time. This proves that the $1|s,A,\text{DIF}|\sum \alpha_jR_j+\sum w_jU_j+bq$ problem is strongly $\mathcal{NP}$-hard as well. ■

7.3 Equal Due-date-assignment Costs

In this section, we study the problem with equal due-date-assignment costs, i.e., $\alpha_j=\alpha>0$, $\forall j\in J$, denoted by $1|s,A,\text{DIF}|\sum \alpha R_j+\sum w_jU_j+bq$.

7.3.1 $\mathcal{NP}$-hardness

Theorem 7.3.1 *The* $1|s,A,DIF|\sum \alpha R_j+\sum w_jU_j+bq$ *problem is* $\mathcal{NP}$*-hard.*

Proof. We prove that the problem is $\mathcal{NP}$-hard by showing its equivalence to the knapsack problem, which is well-known to be $\mathcal{NP}$-hard. Consider an arbitrary instance of the knapsack problem with a set of items $J = \{1, ..., n\}$, each item j has an integral size $0 < p'_j < A'$ and weight $w'_j > 0$, where an integer $A' < \sum_{j=1}^{n} p'_j$ is the size of the knapsack. Our goal is to

$$\text{maximize} \sum_{j \in \Phi} w'_j, \text{ such that: } \sum_{j \in \Phi} p'_j \leq A', \Phi \subseteq J. \tag{7.3.4}$$

Construct an instance of the $1|s, A, \text{DIF}| \sum \alpha R_j + \sum w_j U_j + bq$ problem, by letting $A = A'$, $p_j = p'_j$, $w_j = w'_j$, $\alpha > \sum_{j=1}^{n} w_j$, $s = 0$ and $q = 0$. Since $\alpha > \sum_{j=1}^{n} w_j$, no job would be assigned a due date greater than A in any optimal schedule. Then from equation (7.2.3), we know that all jobs in an optimal schedule have equal assigned due dates A. Moreover, jobs scheduled before A are early and maximize their total tardy weight. Therefore, let the set of early jobs (with $C_j \leq A$) in an optimal schedule for the above instance of the $1|s, A, \text{DIF}| \sum \alpha R_j + \sum w_j U_j + bq$ problem be Φ. Then Φ is also an optimal solution to the above knapsack problem. ■

7.3.2 Pseudo-polynomial Algorithm

Before we propose a pseudo-polynomial algorithm for the $1|s, A, \text{DIF}| \sum \alpha R_j + \sum w_j U_j + bq$ problem, we first make two important observations.

Proposition 7.3.1 *There is an optimal schedule for the* $1|s, A, DIF| \sum \alpha R_j + \sum w_j U_j + bq$ *problem in which all tardy jobs are delivered at the end in a single batch either by themselves or together with some early jobs.*

Proof. Scheduling tardy jobs at the end in the last batch may improve the schedule for early jobs. ■

Proposition 7.3.2 *For the* $1|s, A, DIF| \sum \alpha R_j + \sum w_j U_j + bq$ *problem, there is an optimal schedule* σ*, in which early jobs are processed in shortest processing time first (*SPT*) order.*

Proof. Suppose that two jobs i and k with $p_i > p_k$ are early in consecutive batches in schedule σ with batch-completion times $C_i(\sigma) < C_k(\sigma)$. By Propositions 7.2.1 and 7.2.2, we can assume that $D_i(\sigma) \in \{A, C_i(\sigma)\}$, $D_k(\sigma) \in \{A, C_k(\sigma)\}$, $C_i(\sigma) \leq \frac{w_i}{\alpha} + A$ and $C_k(\sigma) \leq \frac{w_k}{\alpha} + A$. Let σ' be the 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 clear that all early jobs in σ (excluding job i) stay early after the exchange. If $C_i(\sigma') \leq \frac{w_i}{\alpha} + A$, then we are able to let job i be early. In this situation, the due-date-assignment cost will not increase because job i and job k have the same α. If $C_i(\sigma') > \frac{w_i}{\alpha} + A$, then it is best to set $D_i(\sigma') = A$ and let job i be tardy. In this situation, the cost will be reduced by at least

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

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

For the rest of this subsection, let all jobs be indexed so that $p_1 \leq p_2 \leq ... \leq p_n$. We call a batch *delivered*, if the job due dates have been assigned in the batch. On the other hand, we call a batch *pending*, if no job due date has been assigned yet in the batch. Suppose that there is an optimal schedule in which there are exactly m batches, $1 \leq m \leq n$. Then this optimal schedule has makespan $P + ms$ and batch-delivery cost mq. By Proposition 7.3.1, we know that all tardy jobs are in the m-th batch and any (tardy) job i in this batch has a batch completion time $C_i = P + ms$. By Proposition 7.2.2, any tardy job i must satisfy $\frac{w_i}{\alpha} + A < P + ms$. We use dynamic programming in order to find such an optimal schedule. The algorithm iteratively constructs schedules with delivered batches and at most one pending batch (the last one opened) on early jobs, while scheduling some other jobs to be tardy in the m-th

batch. Let (j, l, k, t, x) be the state for a partial schedule on job set $\{1, 2, ..., j\}$, where l is the number of delivered batches, k is the number of jobs in the pending batch and t is the total processing time of the early jobs in the delivered and pending batches. Let x be the sum of the due-date-assignment cost of early jobs in the delivered batches plus the tardy weights of tardy jobs in the m-th batch and the total batch-delivery cost mq. Note that x does not include the due-date-assignment costs of the k jobs in the pending batch since their due dates have not been assigned yet. Since x will include all batch-delivery costs for the m batches from the outset, no delivery cost needs to be added while building the batches. Then the makespan of the delivered and pending batches, including the corresponding batch-setup times, is $t + (l + 1)s$.

Let us first consider the states we can generate based on a partial schedule $(j - 1, l, k, t, x)$ with $j = 1, ..., n$, when aiming at an optimal schedule with exactly m batches. We can do at most the following three operations: (1) Designate job j in the pending batch; (2) Deliver the jobs in the pending batch and start a new pending batch with designated job j in it; (3) Schedule job j in the m-th batch as a tardy job (if $\frac{w_j}{\alpha} + A < P + ms$). In Operation (1), cost x does not need to change. In Operation (2), we assign a common due date $\max\{A, t + ls + s\}$ to the jobs in the pending batch, which makes them early, and we need to add the due-date-assignment cost $\alpha k \max\{t + ls + s - A, 0\}$ to cost x. In Operation (3), since $\frac{w_i}{\alpha} + A < P + ms$, we can assign a due date A to job j, which makes it tardy in the m-th batch, and we need to add the cost w_j to cost x.

Consider now how we can simply complete into a full schedule a partial schedule, $(j - 1, m - 2, k, t, x)$, $j = 1, ..., n$, with exactly $m - 1$ batches ($m - 2$ delivered and one pending), when aiming at an optimal schedule with exactly m batches: We can schedule all remaining jobs $\{j, ..., n\}$ in the m-th batch. For the jobs in the pending batch, we need to assign them the common due date $\max\{A, t + (m - 1)s\}$ making them early, and the due-date-assignment cost of the pending batch, $\alpha k \max\{t + (m - 1)s - A, 0\}$ needs to be added to cost x. For jobs $\{j, ..., n\}$, let

$$J(j, m) = \{i | \frac{w_i}{\alpha} + A < P + ms, i \in \{j, ..., n\}\}. \tag{7.3.5}$$

$J(j,m) \subseteq \{j,...,n\}$ contains exactly those remaining jobs which would be tardy in the m-th batch. If $J(j,m) \neq \emptyset$, then we need to assign a common due date A to the jobs in $J(j,m)$, which makes them tardy in the m-th batch. The tardiness penalties for $J(j,m)$ are $\sum_{i \in J(j,m)} w_i$. Moreover, we need to assign the common due date $P+ms$ to the jobs in $\{j,...,n\} \backslash J(j,m)$, which makes them early in the m-th batch with due-date-assignment costs $\alpha k(j,m) \max\{P+ms-A,0\}$, where

$$k(j,m) = n - j + 1 - |J(j,m)| \tag{7.3.6}$$

is the resulting number of early jobs in the m-th batch. Therefore, the total cost of jobs $\{j,...,n\}$ is

$$x(j,m) = \alpha k(j,m) \max\{P+ms-A,0\} + \sum_{i \in J(j,m)} w_i. \tag{7.3.7}$$

Note that it is possible that there may be some jobs from $\{1,...,j-1\}$ which were previously scheduled to be tardy in the m-th batch, however, their cost would have been added into cost x at the time of their scheduling.

Now let us consider how to reduce the state space. For any two states (j,l,k,t,x_1) and (j,l,k,t,x_2) with $x_1 < x_2$, we can eliminate the second one, because any later states generated from it can not lead to a smaller x value than the value of similar states generated from the first one.

Remark 7.3.1 *For all states with the same entries: $(j,l,k,t,\cdot)$, we only need to keep one of them, which has the smallest x value.*

Algorithm CH7-A1 starts from an empty schedule (state), $(0,0,0,0,mq)$, $m = 1,...,n$, where mq is corresponding to the batch-delivery cost for the m batches and returns a schedule which has the smallest cost among all schedules with exactly m batches. Let $(n,m,0,P,\infty)$ represent an initial fictitious full schedule with exactly m batches and total cost ∞. Then this state will be repeatedly updated by states representing full schedules with smaller total cost x found by the algorithm. Partial schedules for the first j jobs $\{1,2,...,j\}$ are included in set $\mathcal{S}^{(j)}$, $j = 1,...,n$. In

particular, we initialize $\mathcal{S}^{(0)} = \{(0,0,0,0,mq)\}$. At the beginning, we set $\mathcal{T}^* = \{(n,m,0,P,\infty)|\ m = 1,..,n\}$ and at the end $\mathcal{T}^*$ will store the best schedules with exactly $m = 1,...,n$ batches for each m. At the end, Algorithm CH7-A1 finds the optimal solution value, determined by $x^* = \min_{m=1,...,n}\{\theta_m\}$, where $\theta_m = x$ represents the corresponding cost of the best full schedule $(n,m,0,P,x)$ and then traces this back to obtain an optimal schedule and corresponding assigned due dates.

Algorithm CH7-A1

[CandidateSet] Initialize $\mathcal{T}^* = \{(n,1,0,P,\infty),(n,2,0,P,\infty),...,(n,n,0,P,\infty)\}$.

For $m = 1$ **to** n */*Search for an optimal schedule with exactly m batches.*

[Initialization] Set $\mathcal{S}^{(0)} = \{(0,0,0,0,mq)\}$ and $\mathcal{S}^{(j)} = \emptyset$, $j = 1,...,n$.

[Generation] Generate set $\mathcal{S}^{(j)}$ from $\mathcal{S}^{(j-1)}$.

For $j = 1$ **to** n

[Setup] Set $\mathcal{T} = \emptyset$.

For each state $(j-1,l,k,t,x)$ in $\mathcal{S}^{(j-1)}$

1. [FurtherGeneration] Do the following three operations:

   Alternative 1: If $0 < l < m-1$, then set $\mathcal{T} \leftarrow \mathcal{T} \cup (j,l,k+1,t+p_j,x)$ */*Designate job j to be early in the current pending batch.*

   Alternative 2: If $0 \leq l < m-2$, then set $\mathcal{T} \leftarrow \mathcal{T} \cup (j,l+1,1,t+p_j,x+\alpha k \max\{t+(l+1)s-A,0\})$ */*Deliver the jobs in the $(l+1)$-th batch and assign to them the common due date* $\max\{A,t+(l+1)s\}$. *Start a new current batch (batch $(l+2)$) and designate job j to be early in the newly started batch.*

   Alternative 3: If $0 \leq l < m-1$ and $\frac{w_j}{\alpha} + A < P + ms$, then set $\mathcal{T} \leftarrow \mathcal{T} \cup (j,l,k,t,x+w_j)$ */*Schedule job j tardy in the m-th batch.*

2. [SimpleCompletion] If $l = m-2$, then calculate $J(j,m)$, $k(j,m)$ and $x(j,m)$ by equations (7.3.5), (7.3.6) and (7.3.7) and update:

If $\theta_m > x + \alpha k \max\{t + (l+1)s - A, 0\} + x(j,m)$, then let $\theta_m = x + \alpha k \max\{t + (l+1)s - A, 0\} + x(j,m)$ *and* replace the state starting with $(n, m, \cdot, \cdot, \cdot)$ in $\mathcal{T}^*$ by the new best state $(n, m, 0, P, \theta_m)$. */*Complete the partial schedule into a full schedule by delivering the current pending batch (batch $(m-1)$) with assigned common due date* $\max\{A, t + (l+1)s\}$ *for the jobs in it and scheduling jobs* $\{j, ..., n\}$ *into the m-th batch with a common due date* $P + ms$ *for early jobs and a common due date* A *for tardy jobs.*

Endfor

[Elimination] If $j < n$, then for any two states (j, l, k, t, x) and (j, l, k, t, x') with $x < x'$ eliminate from $\mathcal{T}$ the one with x' (Remark 7.3.1).

[Updating] Set $\mathcal{S}^{(j)} = \mathcal{T}$.

Endfor

Endfor

[Optimization] Select the state with the smallest θ_m from $\mathcal{T}^*$ and trace back its ancestors to obtain an optimal schedule and corresponding assigned due dates.

Theorem 7.3.2 *For the* $1|s, A, DIF|\sum \alpha R_j + \sum w_j U_j + bq$ *problem, Algorithm CH7-A1 finds an optimal schedule in* $O(n^4 P)$ *time, where* $P = \sum_{j=1}^{n} p_j$. *Thus the problem is* $\mathcal{NP}$*-hard only in the ordinary sense.*

Proof. As discussed above, the algorithm considers the three alternatives available for adding a job j to a partial schedule represented by state $(j-1, l, k, t, x)$ in $\mathcal{S}^{(j-1)}$. We note that we can only designate job j to be early in alternatives 1 and 2, since we do not know at this point its batch completion time, which is to become its assigned due date. Thus it is possible that when the pending batch containing j finally gets delivered, its assigned due date will be greater than $A + w_j/\alpha$. In this case, it will cost less to make job j tardy by assigning to it the due date A (see Proposition 7.2.1) This lower-cost solution will be generated in alternative 3 and will dominate the schedules

generated under alternatives 1 and 2 in this case. (Thus these higher-cost solutions will be eliminated in the [Elimination] step.)

Let us consider now the complexity of the algorithm. Each state $(j-1,l,k,t,x)$ in $\mathcal{S}^{(j-1)}$ gives rise to at most three alternative states containing j. The upper bound for the number of triplets $\{j,l,k\}$ is n^3. For each $\{j,l,k\}$, there are at most $P+1$ pairs, $\{t,x\}$, because of [Elimination]. Algorithm CH7-A1 runs the outer loop m times, which is upper-bounded by n. Therefore, the overall time complexity is $O(n^4P)$. ■

Corollary 7.3.1 *For the* $1|s,A,DIF|\sum \alpha R_j+\sum w_jU_j+bq$ *problem, if all processing times are equal, i.e.,* $p_j=p>0, \forall j\in J$, *Algorithm CH7-A1 finds an optimal schedule in* $O(n^5)$ *time.*

Proof. Since $p_j=p$, for each $\{j,l,k\}$, there are at most n possible values for t and therefore there are at most n pairs, $\{t,x\}$. By the proof of Theorem 7.3.2, the time complexity is $O(n^5)$. ■

7.3.3 Fully Polynomial Time Approximation Scheme

In this subsection, we show how we can obtain an FPTAS for the $1|s,A,\text{DIF}|\sum \alpha R_j+\sum w_jU_j+bq$ problem.

Step I: Bounds Analysis

Consider a series of *restricted* problems, denoted by $1|s,A,i,\text{DIF}|\sum \alpha R_j+\sum w_jU_j+bq$, $i=0,1,...,n$. These restricted problems are designed in such a way that any feasible schedule for these restricted problems is also a feasible schedule for the $1|s,A,\text{DIF}|\sum \alpha R_j+\sum w_jU_j+bq$ problem. (We have to consider these restricted problems because it is not possible to obtain a good lower bound for the cost of an optimal schedule for the original problem, as we may have even $\sum w_jU_j=0$ for this schedule.) Any feasible schedule for the i-th restricted problem has to satisfy the following two conditions:

Condition 7.3.1 *In any feasible schedule for the* $1|s, A, i, DIF|\sum \alpha R_j + \sum w_j U_j + bq$ *problem, job* i *is a tardy job and has the largest tardiness penalty among tardy jobs. (In particular, if* $i = 0$*, then no job is tardy in any feasible schedule.)*

Condition 7.3.2 *In any feasible schedule for the* $1|s, A, i, DIF|\sum \alpha R_j + \sum w_j U_j + bq$ *problem, all early jobs are in SPT order in order to satisfy Proposition 7.3.2.*

Let v_i^* be the objective value of an optimal schedule σ_i for the i-th restricted problem, $1|s, A, i, \text{DIF}|\sum \alpha R_j + \sum w_j U_j + bq$. Then we have $v_i^* = \sum_{j=1}^{n}[\alpha R_j(\sigma_i) + w_j U_j(\sigma_i)] + b(\sigma_i)q$ by definition. Since $1 \leq b(\sigma_i) \leq n$, we know that $q \leq b(\sigma_i)q \leq nq$. Now the question is how to bound the cost $\sum_{j=1}^{n}[\alpha R_j(\sigma_i) + w_j U_j(\sigma_i)]$. Notice that job i is tardy in schedule σ_i and any job j which is tardy has $w_j \leq w_i$ by Condition 7.3.1. Consider an *auxiliary problem* of minimizing the total due-date-assignment costs, denoted by $1|s, A, J(i), \text{DIF}|\sum \alpha R_j$, on job set $J(i) = \{j | w_j > w_i, j \in J \backslash \{i\}\}$, for which any feasible schedule has to satisfy the following additional condition.

Condition 7.3.3 *In any feasible schedule, all jobs are early and in SPT order and have assigned due dates no greater than* $\frac{w_j}{\alpha} + A$, $\forall j \in J(i)$.

In particular, if $i = 0$, let $w_i = 0$ and $J(0) = \{j | w_j \geq w_i, j \in J\} = J$, i.e., all jobs must be early and satisfy Condition 7.3.3. If there is no feasible schedule for a $J(i)$, let $w^*(i) = \infty$, where $w^*(i)$ denotes the optimal solution value for the $1|s, A, J(i), \text{DIF}|\sum \alpha R_j$ problem. This implies that there is no feasible schedule for the $1|s, A, i, \text{DIF}|\sum \alpha R_j + \sum w_j U_j + bq$ problem either. Therefore, we set the optimal solution value $v_i^* = \infty$ in this case.

We note that by Proposition 7.2.1, every early job in the same batch must be assigned the same due date, which is the larger of A and the batch completion time. Thus every early job in the same batch will share the same batch due date, which by Proposition 7.2.2 and Condition 7.3.3 cannot exceed $\frac{w_j}{\alpha} + A$ for any job in the batch. Suppose that $\sigma(i)$ is an optimal schedule for the $1|s, A, J(i), \text{DIF}|\sum \alpha R_j$ problem with an objective value $w^*(i) < \infty$, then scheduling jobs in $J \backslash J(i)$ after $\sigma(i)$ generates

a schedule, say $\sigma'(i)$, which is feasible for the $1|s, A, i, \text{DIF}|\sum \alpha R_j + \sum w_j U_j + bq$ problem. Consider a job $k \in J \backslash J(i)$ in $\sigma'(i)$. If $C_k(\sigma'(i)) \leq \frac{w_k}{\alpha} + A$, then by Proposition 7.2.1, we know that $D_k(\sigma'(i)) \in \{C_k(\sigma'(i)), A\}$ and job k is early. The due-date-assignment cost and then the final cost for such a k is,

$$\alpha_k \max\{D_k(\sigma'(i)) - A, 0\} \leq \alpha_k \max\{C_k(\sigma'(i)) - A, 0\} \leq w_k \leq w_i. \tag{7.3.8}$$

If $C_k(\sigma'(i)) > \frac{w_k}{\alpha} + A$, then by Proposition 7.3.1, we can set $D_k(\sigma'(i)) = A$ and job k will be tardy. The cost is just w_k. Therefore, the final cost of the schedule $\sigma'(i)$ is at most $w^*(i) + \sum_{j \in J \backslash J(i)} w_j + nq$. Then, by the definition of $J(i)$, we have an upper bound,

$$v_i^* \leq w^*(i) + \sum_{j \in J \backslash J(i)} w_j + nq \leq w^*(i) + n(w_i + q). \tag{7.3.9}$$

Since job i is tardy and there is at least one batch in any optimal schedule, we have

$$v_i^* \geq w^*(i) + w_i + q. \tag{7.3.10}$$

In order to estimate $w^*(i)$, let us define the *batch due-date-assignment cost* $R_j^{(b)}$, which is calculated for *each* batch as the sum of the due-date-assignment costs of the jobs in the batch. Based on the same data as in the $1|s, A, J(i), \text{DIF}|\sum \alpha R_j$ problem, we define a *second auxiliary problem*, denoted by $1|s, A, J(i), \text{DIF}|\alpha \max R_j^{(b)}$, where any feasible schedule has to satisfy Condition 7.3.3 and the goal is to minimize the maximum batch due-date-assignment cost over the batches. Let u_i^* be its optimal solution value. Since there is at least one batch and at most n batches, we know that

$$u_i^* \leq w^*(i) \leq n u_i^*. \tag{7.3.11}$$

Thus, combining equations (7.3.9), (7.3.10) and (7.3.11), we have

$$u_i^* + w_i + q \leq v_i^* \leq n(u_i^* + w_i + q). \tag{7.3.12}$$

If $J(i) = \emptyset$, then simply let $u_i^* = 0$. If $J(i) = J$, then we know that $i = 0$ and $w_i = 0$. When $u_i^* = \infty$, as mentioned before, we have $v_i^* = w^*(i) = \infty$ as well. Therefore, the above equation (7.3.12) still holds. In summary, we have the following lemma.

Lemma 7.3.1 *Let v_i^* be the optimal solution value for the $1|s, A, i, DIF|\sum \alpha R_j + \sum w_j U_j + bq$ problem, whose feasible schedules have to satisfy Condition 7.3.1 and 7.3.2. Let u_i^* be the optimal solution value for the $1|s, A, J(i), DIF|\alpha \max R_j^{(b)}$ problem, whose feasible schedules also have to satisfy Condition 7.3.3. Then we have $L_i' \leq v_i^* \leq nL_i'$, where $L_i' = u_i^* + w_i + q$, $i = 0, 1, ..., n$ and $w_0 = 0$.*

Step II: Initial Bounds

Now let us solve the second auxiliary problem, $1|s, A, J(i)$,DIF$|\ \alpha \max R_j^{(b)}$, to obtain u_i^*. Assume w.l.o.g. that all jobs in $J(i)$ are indexed so that $p_1 \leq ... \leq p_r$, $r = |J(i)|$. Let (j, l, k, d, y) represent a partial schedule on job set $\{1, ..., j\} \subseteq J(i)$, with y representing the largest batch due-date-assignment cost over the jobs in the delivered batches in the partial schedule so far. We also use notation $y(j, l, k, d)$ to emphasize that it is the value corresponding to the state with entries $\{j, l, k, d\}$. As before, l is the number of delivered batches and k is the number of jobs in the current batch, which is pending. If job h has the the smallest $\frac{w_h}{\alpha} + A$ in the current batch, then let $d = \frac{w_h}{\alpha} + A$ be the smallest upper bound for the due date of early jobs in the current batch. When the current batch is delivered, we have to make sure that its completion time (batch due date) is not greater than d in order to satisfy Condition 7.3.3. We set $d = \infty$, if a schedule is empty (infeasible). To schedule job j in a new batch, the current batch l will be delivered at

$$t(j-1, l+1) = \sum_{g=1}^{j-1} p_g + (l+1)s, \tag{7.3.13}$$

which is the makespan of the partial schedule on $\{1, 2, ..., j-1\}$. If $t(j-1, l+1) > d$, then Condition 7.3.3 is violated, i.e., the pending batch cannot be delivered on time and no new schedule can be generated from $(j-1, l, k, d, y)$; otherwise the common due date $\max\{A, t(j-1, l+1)\}$ is assigned to the jobs in the $(l+1)$-th batch with batch due-date-assignment cost $ky(j-1, l+1)$, where

$$y(j-1, l+1) = \alpha \max\{t(j-1, l+1) - A, 0\}. \tag{7.3.14}$$

Let

$$d_j = \frac{w_j}{\alpha} + A \text{ for } j = 1, ..., r \tag{7.3.15}$$

in the following Algorithm CH7-A2(i), which starts from empty schedule $(0, 0, 0, \infty, 0)$. Partial schedules for the first j jobs $\{1, 2, ..., j\}$ are included in set $\mathcal{S}^{(j)}$, $j = 1, ..., n$. In particular, $\mathcal{S}^{(0)} = \{(0, 0, 0, \infty, 0)\}$.

Remark 7.3.2 *For any two states (j, l, k, d, y_1) and (j, l, k, d, y_2) with $y_1 < y_2$, we can eliminate the second one, because any later states generated from it can not lead to a smaller y value than the value of similar states generated from the first one.*

Algorithm CH7-A2(i)

[Initialization] Determine $J(i) = \{j | w_j \geq w_i, j \in J \backslash \{i\}\}$ and $r = |J(i)|$, set $\mathcal{S}^{(0)} = \{(0, 0, 0, \infty, 0)\}$ and $\mathcal{S}^{(j)} = \emptyset$, $j = 1, ..., r$.

[Generation] Generate set $\mathcal{S}^{(j)}$ from $\mathcal{S}^{(j-1)}$.

For $j = 1$ **to** r

[Setup] Set $\mathcal{T} = \emptyset$.

For each $(j-1, l, k, d, y) \in \mathcal{S}^{(j-1)}$

Calculation: Calculate d_j, $y(j-1, l+1)$, $t(j, l+1)$ and $t(j, l+2)$ by equation (7.3.15), (7.3.14) and (7.3.13), respectively.

Alternative 1: If $t(j, l+1) \leq \min\{d, d_j\}$, then set $\mathcal{T} = \mathcal{T} \cup (j, l, k+1, \min\{d, d_j\}, y\}$ */*Schedule job j early in the current batch.*

Alternative 2: If $t(j, l+1) > d$ and $t(j, l+2) \leq d_j$, then let $y' = \max\{y(j-1, l, k, d), ky(j-1, l+1)\}$ and set $\mathcal{T} = \mathcal{T} \cup (j, l+1, 1, d_j, y')$ */*Scheduling job j in the current pending batch would make some jobs in this batch tardy, so schedule job j early in a new pending batch and deliver the $(l+1)$-th batch by assigning the common due date $t(j-1, l+1)$ to the k jobs in it,* where y' represents the largest batch-due-date-assignment cost over the first $l+1$ batches.

Alternative 3: If $t(j,l+2) > d_j$ and $t(j,l+1) > \min\{d, d_j\}$, then go to [Result] */*No feasible schedules are based on this partial schedule, because job j can not be scheduled as an early job in either the current batch ($t(j,l+1) > \min\{d,d_j\}$) or a new batch ($t(j,l+2) > d_j$).*

Endfor

[Elimination] If $j < r$, then for any two states (j,l,k,d,y) and (j,l,k,d,y') with $y < y'$, eliminate from $\mathcal{T}$ the one with y' */* Remark 7.3.2.*

[Updating] Set $\mathcal{S}^{(j)} = \mathcal{T}$.

Endfor

[Result] Set optimal solution value u_i^* equal to the smallest y among all states in $\mathcal{S}^{(r)}$ (If $\mathcal{S}^{(r)} = \emptyset$, then $u_i^* = \infty$).

Theorem 7.3.3 *For the $1|s, A, J(i), DIF|\alpha \max R_j^{(b)}$ problem, whose feasible schedules have to satisfy Condition 7.3.3, Algorithm CH7-A2(i) finds an optimal solution in $O(n^4)$ time.*

Proof. The correctness of the algorithm follows from the discussion preceding it and the observation that it is advantageous for both the earliness requirement and the batch due-date-assignment cost to use as few batches as possible. For each $(j-1,l,k,d,y)$, there are at most three operations. Since $(j,l,k,d,\cdot)$ always holds the smallest y value and d takes at most $n+1$ values, there are at most $O(n^3)$ states in each $\mathcal{S}^{(j)}$ (k is just a work variable recording the number of jobs in the current pending batch). Since there are at most n iterations, the time complexity is $O(n^4)$ indeed. ■

Corollary 7.3.2 *Suppose v_i^* is the optimal solution value for the i-th restricted problem, whose feasible schedules have to satisfy Condition 7.3.1 and 7.3.2. Then Algorithm CH7-A2(i) finds initial bounds such that $L_i' \le v_i^* \le nL_i'$, where $L_i' = u_i^* + w_i + q$, in $O(n^4)$ time.*

Proof. The corollary directly follows from Lemma 7.3.1 and Theorem 7.3.3. ■

Let v^* be the optimal solution value for the original $1|s, A,\text{DIF}|\sum \alpha R_j + \sum w_j U_j + bq$ problem. Then first we know that v^* must fall into one of non-empty intervals: $[L'_i, nL'_i]$, $i = 0, ..., n$. This implies $v^* \geq v'$, where $v' = \min_{i=0,...,n}\{L'_i\}$. Because $v^* \leq \min_{i=0,...,n}\{v^*_i\}$, so $v^* \leq \min_{i=0,...,n}\{nL'_i\} = nv'$. Therefore, we have the following remark.

Remark 7.3.3 *In $O(n^5)$ time, we can determine a pair of initial bounds for v^* such that $v^* \in [v', nv']$, where $v' = \min_{i=0,...,n}\{L'_i\}$.*

Step III: Tight Bounds

In order to narrow the bounds $[v', nv']$, we first present the following Algorithm CH7-A3(u, ε), which is similar to Algorithm CH7-A1 and uses the same state representation (j, l, k, t, x), but also includes interval partitioning for the objective, introduced by Sahni [1976]. Given a target value $u > 0$ for the unknown v^* and an arbitrary small $\varepsilon > 0$, Algorithm CH7-A3(u, ε) is a $(1 - \varepsilon)$-relaxed procedure that reports either $v^* > (1 - \varepsilon)u$ or $v^* \leq u$. Similarly to Remark 7.3.1, we have the following remark.

Remark 7.3.4 *For all states with the same entries: $(j, l, k, \cdot, x)$, we only need to keep the one which has the smallest t value.*

Algorithm CH7-A3(u, ε)

[CandidateSet] Initialize $\mathcal{T}^* = \{(n, 1, 0, P, \infty), (n, 2, 0, P, \infty), ..., (n, n, 0, P, \infty)\}$.

For $m = 1$ **to** n /*Search for an optimal schedule with exactly m batches.

[Initialization] Set $\mathcal{S}^{(0)} = \{(0, 0, 0, 0, mq)\}$ and $\mathcal{S}^{(j)} = \emptyset$, $j = 1, ..., n$.

[Partitioning] Partition the interval $[0, u]$ into $\lceil n/\varepsilon \rceil$ equal subintervals of size $\varepsilon u/n$, with the last one possibly smaller.

[Generation] Generate set $\mathcal{S}^{(j)}$ from $\mathcal{S}^{(j-1)}$.

For $j = 1$ **to** n

[Setup] Set $\mathcal{T} = \emptyset$.

For each state $(j-1, l, k, t, x)$ in $\mathcal{S}^{(j-1)}$

Do the same as in Algorithm CH7-A1.

Endfor

[Elimination] If $j < n$, do the following:

1. If $x > u$, then eliminate from $\mathcal{T}$ any newly generated corresponding state (j, l, k, t, x).
2. For any two states (j, l, k, t, x) and (j, l, k, t', x) with $t \leq t'$, eliminate the one with t' from set $\mathcal{T}$ based on Remark 7.3.4.
3. For states (j, l, k, t, x) with values x falling into the same subinterval, keep only the one with the smallest t value for each subinterval.

[Updating] Set $\mathcal{S}^{(j)} = \mathcal{T}$.

Endfor

Endfor

[Report] If $\mathcal{T}^* = \emptyset$, then report $v^* > (1-\varepsilon)\, u$; otherwise, report $v^* \leq u$ and demonstrate it by taking the solution with the lowest x value from $\mathcal{T}^*$.

Theorem 7.3.4 *In $O(n^5/\varepsilon)$ time, Algorithm CH7-A3(u, ε) either establishes that $v^* > (1-\varepsilon)\, u$ or demonstrates that $v^* \leq u$ (by finding a solution with $x \leq u$), where v^* is the optimal solution value for the $1|s, A, DIF|\sum \alpha R_j + \sum w_j U_j + bq$ problem.*

Proof. Similarly to Theorem 4.4.3, if $\mathcal{T}^* \neq \emptyset$, then there is at least one state (n, l, k, t, x) that has not been eliminated. Therefore, $v^* \leq x \leq u$. If $\mathcal{T}^* = \emptyset$, then, by [Partitioning] and [Elimination], the error introduced in each generation is at most $\varepsilon u/n$. Thus the overall error is at most εu. Therefore, $x > (1-\varepsilon)u$. Because v^* is the smallest value of all possible values x, we deduce $v^* > (1-\varepsilon)u$. For each triplet

$\{j, l, k\}$, there are at most $O(\lceil n/\varepsilon \rceil)$ pairs, $\{t, x\}$. By the proof of Theorem 7.3.2, the running time is then $O(n^5/\varepsilon)$. ■

Next, we consider to use $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ introduced in Chapter 3 to narrow the bounds $[v', nv']$. Let $a_1 = v'$, $a_2 = nv'$, $\omega = u$ and $\mathcal{P}$ be the $1|s, A, \text{DIF}|\sum \alpha R_j + \sum w_j U_j + bq$ problem. Then Algorithm CH7-A3$(u, 1/3)$ can be used as $\mathcal{A}(\omega, \mathcal{P})$. Since Algorithm CH7-A3$(u, 1/3)$ runs in $O(n^5)$ time, then in $O(n^5 \log\log n)$ time, $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ reports ξ that implies a pair of tight bounds $[\xi, 3\xi]$. In this situation, let us refer to $\mathcal{BIP}[a_1, a_2, \mathcal{A}(\omega, \mathcal{P})]$ as Algorithm CH7-A4.

Corollary 7.3.3 *Algorithm CH7-A4 can narrow the bounds* $[v', nv']$ *into* $[\bar{v}, 3\bar{v}]$ *in* $O(n^5 \log\log n)$ *time by setting* $\bar{v} = \xi$.

Step IV: Approximation

For any given $\varepsilon > 0$, using the bounds $\bar{v} \leq v^* \leq 3\bar{v}$ obtained by Algorithm CH7-A4, we run a slightly changed version of Algorithm CH7-A3(u, ε), called Algorithm CH7-A5, with $u = (1 + \varepsilon/3)3\bar{v}$: The only difference is that in the [Partitioning] step we partition $[0, u] = [0, (1 + \varepsilon/3)3\bar{v}]$ into $n\lceil 3/\varepsilon + 1 \rceil$ intervals of size at most $\varepsilon\bar{v}/n$, so the cumulative error over n iterations will be no more than $\varepsilon\bar{v}$. Since we know that the problem has an optimal solution value $v^* \leq 3\bar{v}$, the algorithm will find a solution v for the $1|s, A, \text{DIF}|\sum \alpha R_j + \sum w_j U_j + bq$ problem such that $v \leq 3\bar{v} + \varepsilon\bar{v} = u$, which means that the algorithm will never end with an empty $\mathcal{T}^*$. Furthermore, whichever subinterval of $[\bar{v}, 3\bar{v}]$ v^* falls into, the algorithm will generate an approximate solution v with at most $\varepsilon\bar{v}$ error away from it, i.e., $v \leq v^* + \varepsilon\bar{v} \leq (1 + \varepsilon)v^*$. Therefore, we have the following corollary.

Corollary 7.3.4 *For any given* $\varepsilon > 0$, *Algorithm CH7-A5 finds a* $(1+\varepsilon)$*-approximate solution for the* $1|s, A, DIF|\sum \alpha R_j + \sum w_j U_j + bq$ *problem in* $O(n^5/\varepsilon)$ *time.*

Now we are ready to present the final Algorithm CH7-A6, which combines into an FPTAS the previous algorithms as subroutines.

Algorithm CH7-A6

[InitialBounds]: Set $v' = \infty$.

For $i = 0$ **to** n

Run Algorithm CH7-A2(i) and set $v' = \min\{v', u_i^* + q + w_i\}$.

Endfor

[TightBounds]: Run Algorithm CH7-A4 and obtain: $\bar{v} \leq v^* \leq 3\bar{v}$.

[Approximation]: Run Algorithm CH7-A5 and obtain an approximate schedule.

Theorem 7.3.5 *For the* $1|s, A, DIF|\sum \alpha R_j + \sum w_j U_j + bq$ *problem, Algorithm CH7-A6 finds a* $(1+\varepsilon)$*-approximate solution in* $O(n^5/\varepsilon + n^5 \log\log n)$ *time.*

Proof. The complexity and correctness follow from the results for the component algorithms. ■

7.4 Equal Due-date-assignment Costs and Equal Tardiness Penalties

In this section, we propose a polynomial algorithm for the case when the tardiness penalties and due-date-assignment costs are equal, i.e., $w_j = w > 0$ and $\alpha_j = \alpha > 0$, $\forall j \in J$, denoted by $1|s, A, \text{DIF}|\sum \alpha R_j + \sum w U_j + bq$. Note that Proposition 7.2.1, 7.2.2, 7.3.1 and 7.3.2 can still be applied to this problem. Assume that all jobs are indexed so that $p_1 \leq ... \leq p_n$. Before we present the algorithm, let us emphasize the following two important propositions.

Proposition 7.4.1 *There is an optimal schedule for the* $1|s, A, DIF|\sum \alpha R_j + \sum w U_j + bq$ *problem, in which all early jobs are scheduled before or at* $\frac{w}{\alpha} + A$.

Proof. By Proposition 7.2.2, $\frac{w}{\alpha} + A$ is an upper bound for any assigned due date. ■

Proposition 7.4.2 *If there is an optimal schedule with $1 \leq h \leq n$ early jobs for the $1|s, A, DIF|\sum \alpha R_j + \sum wU_j + bq$ problem, then these early jobs are exactly $\{1, ..., h\}$.*

Proof. Proposition 7.4.1 implies there is an optimal schedule where the early jobs are delivered in batches before or at $\frac{w}{\alpha} + A$ and the tardy jobs (if any) are delivered after $\frac{w}{\alpha} + A$ 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 $\alpha(p_i - p_k)$ and can not lead to an increased cost. Exchanging all such pairs will generate an early job set as claimed. ■

Suppose that early jobs form a job set $J_E \subseteq J$. Our subproblem (defined on J_E) is to find a schedule which minimizes the sum of the due-date-assignment costs and the batch-delivery costs, denoted by $1|s, A, J_E, \text{DIF}|\sum \alpha R_j + bq$. If $s = 0$ and $A = 0$, it is equivalent to the $1||\sum \alpha C_j + bq$ problem on the same job set J_E, which can be solved in polynomial time [Hall and Potts, 2003].

Let (j, l, k, z) represent a partial schedule on job set $\{1, ..., j\}$ with cost z, where l is the number of batches (including the current batch), and k is the number of jobs in the current batch, which is pending. Again, we use notation $z = z(j, l, k)$ to emphasize that it is the value corresponding to the state with entries, $\{j, l, k\}$. The cost of the current batch if it gets delivered with job j as its last job, including its batch-delivery cost, is $\hat{z}(j, l, k) = \alpha k \max\{t(j, l) - A, 0\} + q$, where $t(j, l) = \sum_{i=1}^{j} p_i + ls$ is the makespan. As an alternative, we can also schedule all remaining jobs $\{j + 1, ..., n\}$ into the current batch and deliver them together. Let $d = \frac{w}{\alpha} + A$ in Algorithm CH7-A7. Then if $t(n, l) > d$, then all jobs in the current batch will be tardy and the cost is $\bar{z}(j, l, k) = \sum_{i=j-k}^{n} w_i$ and if $t(n, l) \leq d$, then all jobs in the current batch will be early and the cost is $\tilde{z}(j, l, k) = \alpha(k + n - j) \max\{t(n, l) - A, 0\}$.

Algorithm CH7-A7

[Initialization]: Set $\mathcal{S}^{(0)} = \{(0,1,0,q)\}$ and $\mathcal{S}^{(j)} = \{(j,l,k,\infty)\}$, $j,l,k = 1,...,n$ /* *Notice that* $z(j,l,k) = \infty$, $\forall j,l,k$.

[Generation]: Update set $\mathcal{S}^{(1)}$, $\mathcal{S}^{(2)}$, ...,$\mathcal{S}^{(n)}$.

For $j = 1$ **to** n

for each $(j-1,l,k,z) \in \mathcal{S}^{(j-1)}$

1. If $t(j,l) \leq d$, then set $z' = \min\{z(j-1,l,k), z(j,l,k+1)\}$ and update $(j,l,k+1,z')$ /**Schedule job j early in the current batch.*
2. If $t(j,l+1) \leq d$, then set $z' = \min\{z(j-1,l,k) + \hat{z}(j-1,l,k), z(j,l+1,1)\}$ and update $(j,l+1,1,z')$ /**Schedule job j early in a new batch and deliver the l-th batch.*
3. If $t(n,l) \leq d$, then set $z' = \min\{z(j-1,l,k) + \tilde{z}(j-1,l,k), z(n,l,0)\}$ and update $(n,l,0,z')$ /**Schedule jobs $\{j,...,n\}$ early in the current batch and deliver it.*
4. If $t(n,l) > d$, then set $z' = \min\{z(j-1,l,k) + \bar{z}(j-1,k,l), z(n,l,0)\}$ and update $(n,l,0,z')$ /**Schedule jobs $\{j,...,n\}$ tardy in the current batch and deliver it.*

Endfor

Endfor

[Optimization]: Select the state with the smallest z in $\mathcal{S}^{(n)}$ and trace back to obtain an optimal schedule and corresponding assigned due dates.

Theorem 7.4.1 *For the* $1|s,A,DIF|\sum \alpha R_j + \sum wU_j + bq$ *problem, Algorithm CH7-A7 finds an optimal solution in* $O(n^3)$ *time.*

Proof. For each state $(j-1, l, k, z)$, there are at most four operations. Since state $(j, l, k, \cdot)$ always holds the smallest cost and there are at most n batches and n jobs in each batch, there are at most $(n+1)^2$ states in each $\mathcal{S}^{(j)}$. We also know that there are at most n iterations, therefore the time complexity is $O(n^3)$. ■

7.5 Summary

In this chapter, we studied three DIF problems. Firstly, we proved that the problem with arbitrary due-date-assignment costs is strongly $\mathcal{NP}$-hard. Then, we provided a $\mathcal{NP}$-hardness proof and proposed a pseudo-polynomial time algorithm and an FPTAS for the problem with equal due-date-assignment costs. Finally, we found a polynomial algorithm for the problem with equal due-date-assignment costs and equal tardiness penalties.

Chapter 8

Models with SLK and TWK Assignable Due Dates

In Chapters 6 and 7, we have discussed supply chain scheduling problems with delivery costs and type CON and type DIF due date assignment. In this chapter, we extend our efforts to two similar problems, but with type SLK and type TWK assignable due dates, where SLK means that the assigned due dates are the sum of the processing time and a non-negative *slack*, i.e., $D_j(\theta) = p_j + \theta$, $\theta \geq 0$ and TWK means that the assigned due dates are the product of the processing time and a non-negative *coefficient*, i.e., $D_j(\eta) = \eta p_j$, $\eta \geq 0$.

8.1 Introduction

In the SLK and the TWK problems, we need to assign a value to the common slack and the common coefficient variable. Once the value is determined, the problem is reduced to the $1|s|\sum w_jU_j + bq$ problem, which has been well-studied in Chapter 4. Recall that Algorithm CH4-A1 is a pseudo-polynomial algorithm for the $1|s|\sum w_jU_j + bq$ problem and Algorithm CH4-A7 is an FPTAS for the $1|s|\sum w_jU_j + bq$ problem.

8.2 SLK Due Date Assignment

In this section, we first define the SLK problem and prove that it is $\mathcal{NP}$-hard. Then we present a pseudo-polynomial algorithm for it using Algorithm CH4-A1. Finally, we propose an FPTAS for the problem using Algorithm CH4-A7.

8.2.1 Preliminaries

Note that all the terminologies, assumptions and notations introduced in Chapter 3 apply in this chapter. In the SLK problem, let $D_j(\theta) = p_j + \theta \ \forall j \in J$ denote the *assigned due date*, where $\theta \geq 0$ is the slack variable. Then $d_j(\theta) = \max\{D_j(\theta), A_j\}$ is the *acting due date*. Thus the tardiness indicator variable is $U_j(\theta) = 1$ if $C_j > d_j(\theta)$, where C_j is the completion time of job j, and $U_j(\theta) = 0$ otherwise. Our goal is to find a schedule minimizing the sum of the weighted number of tardy jobs, the due-date-assignment costs and the batch-delivery costs, denoted by $1|s, A_j, \text{SLK}|\sum \alpha_j R_j(\theta) + \sum w_j U_j(\theta) + bq$. Next let us prove the $\mathcal{NP}$-hardness using the knapsack problem.

Theorem 8.2.1 *The SLK problem is $\mathcal{NP}$-hard.*

Proof. Consider an instance of the well-known $\mathcal{NP}$-hard knapsack problem:

$$\text{Maximize: } \sum_{j\in\Phi} w_j, \text{ such that } \sum_{j\in\Phi} p_j \leq A, \ \Phi \subseteq J = \{1, ..., n\}. \tag{8.2.1}$$

Construct an instance of the SLK problem, in which $s = 0$, $q = 0$, $\alpha_j \gg w_j$ and $P > A_j = A > p_j$, $\forall j \in J$. Since it is cheaper to assign $\theta = 0$, i.e., $D_j(\theta) = p_j$, $\forall j \in J$, no job would have an assigned due date $D_j(\theta) > A$. Therefore, we have $R_j(\theta) = \max\{D_j(\theta) - A, 0\} = 0$ and $d_j(\theta) = \max\{D_j(\theta), A\} = A$, $\forall j \in J$.

Suppose $\Psi \subseteq J$ is the set that includes all early jobs in an optimal solution to the above SLK instance. Since Ψ also minimizes the total tardiness penalties of $J - \Psi$, setting $\Phi = \Psi$ is also an optimal solution to the above knapsack instance. Therefore, the knapsack problem reduces to our SLK instance. We proved that the SLK problem is $\mathcal{NP}$-hard. ■

8.2.2 Pseudo-polynomial Algorithm

We now introduce a pseudo-polynomial algorithm for solving the SLK problem. In the SLK problem, if we know the value of the slack variable, i.e., $\theta = \bar{\theta} \geq 0$, then we can easily calculate $D_j(\bar{\theta}) = p_j + \bar{\theta}$, $d_j(\bar{\theta}) = \max\{D_j(\bar{\theta}), A_j\}$ and $R_j(\bar{\theta}) = \max\{D_j(\bar{\theta}) - A_j, 0\}$, $\forall j \in J$. In this case, the corresponding due-date-assignment cost is $\sum_{j=1}^{n} \alpha_j R_j(\bar{\theta})$. In order to minimize the total cost, we have to find a schedule minimizing the sum of the weighted number of tardy jobs and the batch-delivery costs with respect to $d_j(\bar{\theta})$, denoted by $1|s, d_j(\bar{\theta})|\sum w_j U_j(\bar{\theta}) + bq$. This is indeed the $1|s|\sum w_j U_j + bq$ problem in which $d_j(\bar{\theta})$ is the given due date of job j, $\forall j \in J$.

Before we describe Algorithm CH8-A1, which finds an optimal solution, we determine an upper bound for $\bar{\theta}$. Since there are at most n batches in any feasible schedule, the largest possible makespan is $P + ns$. This implies that the largest possible value for any meaningful acting due dates is $P + ns$, i.e.,

$$d_{\min}(\bar{\theta}) = \min_{j \in J}\{p_j + \bar{\theta}, A_j\} \leq \min_{j \in J}\{p_j + \bar{\theta}\} = p_{\min} + \bar{\theta} \leq P + ns, \tag{8.2.2}$$

where $p_{\min} = \min_{j \in J}\{p_j\}$. Therefore, we have $\bar{\theta} \leq P + ns - p_{\min}$.

Algorithm CH8-A1

[Initialization]: Set $v^* = \infty$.

For $\bar{\theta} = 0$ **to** $P + ns - p_{\min}$

1. Determine $d_j(\bar{\theta}) = \max\{D_j(\bar{\theta}), A_j\}$ and $R_j(\bar{\theta}) = \max\{D_j(\bar{\theta}) - A_j, 0\}$, where $D_j(\bar{\theta}) = p_j + \bar{\theta}$, $\forall j \in J$.
2. Run Algorithm CH4-A1 for the $1|s, d_j(\bar{\theta})|\sum w_j U_j + bq$ problem and update $v^* = \min\{\sum_{j=1}^{n} \alpha_j R_j(\bar{\theta}) + v(\bar{\theta}), v^*\}$. */*$v(\bar{\theta})$ is the optimal solution value for the $1|s, d_j(\bar{\theta})|\sum w_j U_j + bq$ problem obtained by Algorithm CH4-A1.*

Endfor

[Result]: Trace back v^*, $\bar{\theta}$ and $v(\bar{\theta})$ to obtain the optimal schedule.

Theorem 8.2.2 *Algorithm CH8-A1 is a pseudo-polynomial algorithm, which finds an optimal solution for the SLK problem in* $O(n^3[\min\{P+ns, W+nq\}][P+ns])$ *time and space, where* $P = \sum_{j=1}^{n} p_j$ *and* $W = \sum_{j=1}^{n} w_j$. *This shows that the SLK problem is* $\mathcal{NP}$*-hard only in the ordinary sense.*

Proof. We know that Algorithm CH4-A1 runs in $O(n^3[\min\{d_n(\bar{\theta}), P+ns, W+nq\}])$ time and space for the $1|s, d_j(\bar{\theta})|\sum w_jU_j + bq$ problem by Theorem 4.3.1. Since the largest acting due date considered is $d_n(\bar{\theta}) = p_{\max}+\bar{\theta} = p_{\max}+P+ns-p_{\min} \geq P+ns$, where $p_{\max} = \max_{j\in J}\{p_j\}$, we have $\min\{d_n(\bar{\theta}), P+ns, W+nq\} = \min\{P+ns, W+nq\}$. Moreover, Algorithm CH8-A1 calls Algorithm CH4-A1 at most $O(P+ns-p_{\min}+1) = O(P+ns)$ times. Therefore the complexity of Algorithm CH8-A1 is $O(n^3[\min\{P+ns, W+nq\}][P+ns])$. ■

8.2.3 Fully Polynomial Time Approximation Scheme

Suppose that we have h distinct values $A_j - p_j$, i.e.,

$$0 < A_{[1]} - p_{[1]} < ... < A_{[h]} - p_{[h]} < P + ns - p_{\min}. \tag{8.2.3}$$

The due-date-assignment cost for $\theta = x$ can be represented as

$$\sum_{j=1}^{n} \alpha_j R_j(x) = \sum_{j=1}^{h} \bar{\alpha}_{[j]} R_{[j]}(x) = \sum_{j=1}^{h} \bar{\alpha}_{[j]} \max\{p_{[j]} + x - A_{[j]}, 0\}, \tag{8.2.4}$$

where $\bar{\alpha}_{[j]} = \sum_{i\in\bar{J}_{[j]}} \alpha_i$ and $\bar{J}_{[j]} = \{i | A_i - p_i = A_{[j]} - p_{[j]}, \forall i \in J\}$. This function is a piecewise linear function of x as shown in Figure 8.1, where it is denoted by bold and sloped lines. When x passes a break point in equation (8.2.3) denoted by a circle in Figure 8.1, $\sum_{j=1}^{h} \bar{\alpha}_{[j]} R_{[j]}(x)$ will include one more job set, say job $\bar{J}_{[j]}$, with a positive cost and the slope will increase by $\bar{\alpha}_{[j]}$.

Now let us define $\theta_0 = 0$, $\theta_1 = A_{[1]} - p_{[1]}$, ..., $\theta_h = A_{[h]} - p_{[h]}$ and $\theta_{h+1} = P + ns - p_{\min}$. For any given $\varepsilon > 0$, let us further divide each interval $[\theta_i, \theta_{i+1})$

$(1 \leq i \leq h)$ into l_i smaller intervals:

$$
\begin{aligned}
H_{i,1}^{SLK} &= (\theta_i, \theta_i + \bar{\delta}_i]; \\
H_{i,2}^{SLK} &= (\theta_i + \bar{\delta}_i, \theta_i + \bar{\delta}_i + \bar{\delta}_i(1+\varepsilon)]; \\
&\cdots\cdots \\
H_{i,l_i}^{SLK} &= (\theta_i + \bar{\delta}_i \sum_{k=0}^{l_i-2}(1+\varepsilon)^k, \min\{\theta_{i+1}, \theta_i + \bar{\delta}_i \sum_{k=0}^{l_i-1}(1+\varepsilon)^k\}],
\end{aligned}
\tag{8.2.5}
$$

where l_i is such that

$$
\theta_i + \bar{\delta}_i \sum_{k=0}^{l_i-2}(1+\varepsilon)^k \leq \theta_{i+1} \leq \theta_i + \bar{\delta}_i \sum_{k=0}^{l_i-1}(1+\varepsilon)^k
$$

and thus

$$
\bar{\delta}_i = 1 + \varepsilon + \varepsilon \frac{\sum_{j=1}^{i} \bar{\alpha}_{[j]}(\theta_i - \theta_j)}{\sum_{j=1}^{i} \bar{\alpha}_{[j]}}. \tag{8.2.6}
$$

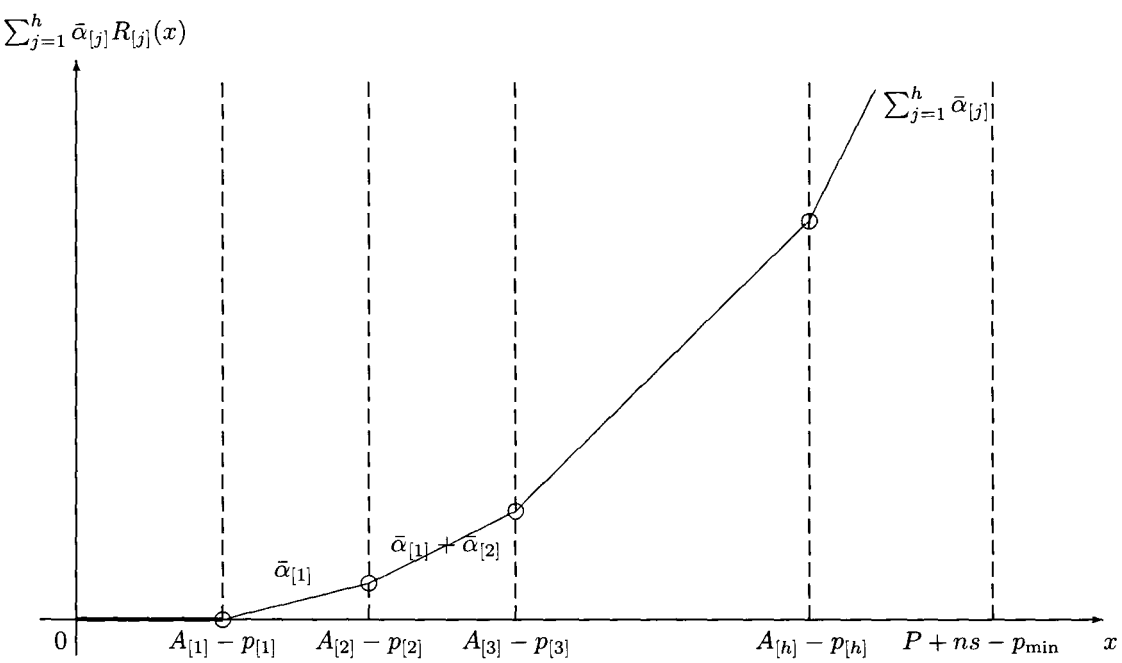


Figure 8.1: The SLK due-date-assignment cost as a piecewise linear function of x.

It is sufficient to chose l_i to be

$$l_i = \lceil \log_{(1+\varepsilon)} \frac{\varepsilon(\theta_{i+1} - \theta_i)}{\bar{\delta}_i} + 1 \rceil. \tag{8.2.7}$$

We note that

$$O(l_i) \leq O(\log \theta_{i+1}) \leq O(\log(P + ns)). \tag{8.2.8}$$

Let $\sigma(\tilde{\theta})$ be an optimal schedule for the SLK problem with the slack value $\theta = \tilde{\theta} \in H_{i,k}^{SLK}$ for some $0 < i \leq h$ and some $0 < k \leq l_i$. Thus the total cost is

$$v^*(\tilde{\theta}) = \sum_{j=1}^{n} \alpha_j R_j(\tilde{\theta}) + \sum_{j=1}^{n} w_j U_j(\sigma(\tilde{\theta})) + b(\sigma(\tilde{\theta}))q, \tag{8.2.9}$$

where

$$\sum_{j=1}^{n} \alpha_j R_j(\tilde{\theta}) = \sum_{j=1}^{h} \bar{\alpha}_{[j]} \max\{p_{[j]} + \tilde{\theta} - A_{[j]}, 0\} = \sum_{j=1}^{i} \bar{\alpha}_{[j]}(\tilde{\theta} - \theta_j). \tag{8.2.10}$$

It is clearly true that schedule $\sigma(\tilde{\theta})$ is also optimal for the $1|s, d_j(\tilde{\theta})| \sum w_j U_j(\tilde{\theta}) + bq$ problem. Let

$$\theta' = \begin{cases} \theta_i + \bar{\delta}_i \sum_{r=0}^{k-1} (1+\varepsilon)^r, & \text{if } 0 < k < l_i \\ \min\{\theta_{i+1}, \theta_i + \bar{\delta}_i \sum_{r=0}^{k-1} (1+\varepsilon)^r\}, & \text{if } k = l_i \end{cases} \tag{8.2.11}$$

be the right end point of the interval $H_{i,k}^{SLK}$. Since $\tilde{\theta} \in H_{i,k}^{SLK}$, then we have $\theta' \geq \tilde{\theta}$ and thus

$$d_j(\theta') = \max\{p_j + \theta', A_j\} \geq d_j(\tilde{\theta}) = \max\{p_j + \tilde{\theta}, A_j\}, \forall j \in J. \tag{8.2.12}$$

In order to estimate the due-date-assignment cost, let us consider two cases: $k = 1$

and $k > 1$. When $k = 1$, we have (Notice that $\tilde{\theta} \in H_{i,k}^{SLK}$ implies $\tilde{\theta} > \theta_i$.)

$$\begin{aligned}
\sum_{j=1}^{n} \alpha_j R_j(\theta') &= \sum_{j=1}^{h} \bar{\alpha}_{[j]} \max\{p_{[j]} + \theta' - A_{[j]}, 0\} \\
&\leq \sum_{j=1}^{i} \bar{\alpha}_{[j]}[\theta_i + \bar{\delta}_i - \theta_j] \\
&= \sum_{j=1}^{i} \bar{\alpha}_{[j]}[\theta_i + 1 + \varepsilon + \varepsilon \frac{\sum_{j=1}^{i} \bar{\alpha}_{[j]}(\theta_i - \theta_j)}{\sum_{j=1}^{i} \bar{\alpha}_{[j]}} - \theta_j] \\
&= (1+\varepsilon) \sum_{j=1}^{i} \bar{\alpha}_{[j]}[\theta_i + 1 - \theta_j] \\
&\leq (1+\varepsilon) \sum_{j=1}^{i} \bar{\alpha}_{[j]}(\tilde{\theta} - \theta_j) = (1+\varepsilon) \sum_{j=1}^{n} \alpha_j R_j(\tilde{\theta}), \qquad (8.2.13)
\end{aligned}$$

where $R_j(\theta') = \max\{p_j + \theta' - A_j, 0\}$ and $R_j(\tilde{\theta}) = \max\{p_j + \tilde{\theta} - A_j, 0\}$, $\forall j \in J$. Similarly, by the design of the smaller intervals in (8.2.5), when $k > 1$, we have $\tilde{\theta} > \theta_i + \bar{\delta}_i \sum_{r=0}^{k-2}(1+\varepsilon)^r$, $1 < k \leq l_i$ and then

$$\begin{aligned}
\sum_{j=1}^{n} \alpha_j R_j(\theta') &= \sum_{j=1}^{h} \bar{\alpha}_{[j]} \max\{p_{[j]} + \theta' - A_{[j]}, 0\} \\
&\leq \sum_{j=1}^{i} \bar{\alpha}_{[j]}[\theta_i + \bar{\delta}_i \sum_{r=0}^{k-1}(1+\varepsilon)^r - \theta_j] \\
&= \sum_{j=1}^{i} \bar{\alpha}_{[j]}(\theta_i - \theta_j) + \sum_{j=1}^{i} \bar{\alpha}_{[j]}[\bar{\delta}_i \sum_{r=1}^{k-1}(1+\varepsilon)^r] + \sum_{j=1}^{i} \bar{\alpha}_{[j]}[\bar{\delta}_i (1+\varepsilon)^0] \\
&= (1+\varepsilon) \sum_{j=1}^{i} \bar{\alpha}_{[j]}(\theta_i - \theta_j) + \sum_{j=1}^{i} \bar{\alpha}_{[j]}[\bar{\delta}_i \sum_{r=1}^{k-1}(1+\varepsilon)^r] + (1+\varepsilon) \sum_{j=1}^{i} \bar{\alpha}_{[j]} \\
&= (1+\varepsilon) \sum_{j=1}^{i} \bar{\alpha}_{[j]}[\theta_i + \bar{\delta}_i \sum_{r=0}^{k-2}(1+\varepsilon)^r + 1 - \theta_j] \\
&\leq (1+\varepsilon) \sum_{j=1}^{i} \bar{\alpha}_{[j]}(\tilde{\theta} - \theta_j) = (1+\varepsilon) \sum_{j=1}^{n} \alpha_j R_j(\tilde{\theta}), \qquad (8.2.14)
\end{aligned}$$

where we substitute $\bar{\delta}_i$ by equation (8.2.6) in the last term of the third line, and the last inequality holds because $\theta_i + \bar{\delta}_i \sum_{r=0}^{k-2}(1+\varepsilon)^r < \tilde{\theta}$.

Let $w_{opt}(\theta')$ be the optimal solution value for the $1|s, d_j(\theta')|\sum w_jU_j(\theta') + bq$ problem. By equation (8.2.12), we know that

$$w_{opt}(\theta') \leq \sum_{j=1}^{n} w_jU_j(\sigma(\tilde{\theta})) + b(\sigma(\tilde{\theta}))q. \tag{8.2.15}$$

Let $\sigma_{apx}(\theta')$ be a schedule for the $1|s, d_j(\theta')|\sum w_jU_j(\theta') + bq$ problem such that its cost

$$w_{apx}(\theta') \leq (1+\varepsilon)w_{opt}(\theta'). \tag{8.2.16}$$

Then the total cost of the SLK problem on schedule $\sigma_{apx}(\theta')$ is

$$\begin{aligned} v(\theta') &= \sum_{j=1}^{n} \alpha_j R_j(\theta') + w_{apx}(\theta') \\ &\leq (1+\varepsilon)\sum_{j=1}^{n} \alpha_j R_j(\tilde{\theta}) + (1+\varepsilon)w_{opt}(\theta') \\ &\leq (1+\varepsilon)\sum_{j=1}^{n} \alpha_j R_j(\tilde{\theta}) + (1+\varepsilon)[\sum_{j=1}^{n} w_jU_j(\sigma(\tilde{\theta})) + b(\sigma(\tilde{\theta}))q] \\ &\leq (1+\varepsilon)v^*(\tilde{\theta}), \end{aligned} \tag{8.2.17}$$

which follows from equations (8.2.13), (8.2.14), (8.2.15) and (8.2.16).

Now we are ready to present an FPTAS for the SLK problem using Algorithm CH4-A7, which is the FPTAS developed for the $1|s|\sum w_jU_j + bq$ problem in Chapter 4. Algorithm CH8-A2 calls Algorithm CH4-A7 for each θ' in equation (8.2.11) to obtain an approximate solution for the $1|s, d_j(\theta')|\sum w_jU_j(\theta') + bq$ problem. For some θ', it may happen that some $D_j(\theta') = p_j + \theta'$ is not an integer. In this situation, we will take the value of $D_j(\theta') = \lfloor p_j + \theta' \rfloor$.

Algorithm CH8-A2

[Initialization]: Set $\mathcal{T} = \emptyset$ and determine h, δ_i, θ_i and l_i for any given $\varepsilon > 0$ based on the above discussion.

For $i = 1$ **to** h

For $k = 1$ **to** l_i

1. Set θ' as in equation (8.2.11).
2. Determine $d_j(\theta') = \max\{D_j(\theta'), A_j\}$ and $R_j(\theta') = \max\{D_j(\theta') - A_j, 0\}$, where $D_j(\theta') = \lfloor p_j + \theta' \rfloor$, $\forall j \in J$.
3. Run Algorithm CH4-A7 for the $1|s, d_j(\theta')|\sum w_j U_j + bq$ problem and obtain $v(\theta')$. */*$v(\theta')$ is the approximate solution value obtained by Algorithm CH4-A7 for the $1|s, d_j(\theta')|\sum w_j U_j + bq$ problem.*
4. Set $\mathcal{T} \leftarrow \mathcal{T} \cup (\theta', \sum_{j=1}^{n} \alpha_j R_j(\theta') + v(\theta'))$.

Endfor

Endfor

[Result]: Select the state with the smallest $\sum_{j=1}^{n} \alpha_j R_j(\theta') + v(\theta')$ in $\mathcal{T}$ and trace back to obtain the schedule.

Theorem 8.2.3 *Algorithm CH8-A2 is an FPTAS, which finds a $(1+\varepsilon)$-approximate solution to the SLK problem in $O(\sum_{i=1}^{h} l_i[n^4 \log\log n + n^4/\varepsilon])$ time for any given $\varepsilon > 0$, where l_i is defined in equation (8.2.7).*

Proof. The correctness of the algorithm follows from the preceding discussion and Theorem 4.4.4. Algorithm CH4-A7 solves the $1|s, d_j(\theta')|\sum w_j U_j + bq$ problem in $O(n^4 \log\log n + n^4/\varepsilon)$ time and Algorithm CH8-A2 calls Algorithm CH4-A7 at most $O(\sum_{i=1}^{h} l_i)$ times. Note that $O(l_i) \leq O(\log(P + ns))$. Therefore, Algorithm CH8-A2 runs in $O(\sum_{i=1}^{h} l_i[n^4 \log\log n + n^4/\varepsilon]) \leq O(\log(P + ns)[n^5 \log\log n + n^5/\varepsilon])$ time. ■

8.3 TWK Due Date Assignment

In this section, we first define the TWK problem in detail and prove its $\mathcal{NP}$-hardness. Then we present a pseudo-polynomial algorithm for it using Algorithm CH4-A1. Finally we develop an FPTAS for it using Algorithm CH4-A7.

8.3.1 Preliminaries

As was the case with the SLK problem, all the terminologies, assumptions and notations introduced in Chapter 3 apply in this chapter too. In the TWK problem, let $D_j(\eta) = \eta p_j \ \forall j \in J$ denote the *assigned due date*, where $\eta \geq 0$ is the coefficient variable, and let $d_j(\eta) = \max\{D_j(\eta), A_j\}$ be the *acting due date*. Similarly, the tardiness indicator variable is $U_j(\eta) = 1$ if $C_j > d_j(\eta)$, where C_j is the completion time of job j, and $U_j(\eta) = 0$ otherwise. Our goal is to find a schedule minimizing the sum of the weighted number of tardy jobs, the due-date-assignment costs and the batch-delivery costs, denoted by $1|s, A_j, \text{TWK}|\sum \alpha_j R_j(\eta) + \sum w_j U_j(\eta) + bq$.

Theorem 8.3.1 *The TWK problem is $\mathcal{NP}$-hard.*

Proof. Consider an instance of the TWK problem, in which $s = 0$, $q = 0$, $\alpha_j \gg w_j$ and $P > A_j = A > p_j$, $\forall j \in J$. Then no job would have an assigned due date $D_j(\eta) > A$. Therefore, we have $R_j(\eta) = \max\{D_j(\eta) - A, 0\} = 0$ and $d_j(\eta) = \max\{D_j(\eta), A\} = A$, $\forall j \in J$. It is not hard to see, by the same reduction as for the SLK problem, that this instance is equivalent to the knapsack problem defined by (8.2.1). ■

8.3.2 Pseudo-polynomial Algorithm

In the TWK problem, if the coefficient variable has been determined, i.e., $\eta = \bar{\eta} \geq 0$, then we can also easily calculate $D_j(\bar{\eta}) = \bar{\eta} p_j$, $d_j(\bar{\eta}) = \max\{D_j(\bar{\eta}), A_j\}$ and $R_j(\bar{\eta}) = \max\{D_j(\bar{\eta}) - A_j, 0\}$, $\forall j \in J$. Moreover, the corresponding due-date-assignment cost is $\sum_{j=1}^{n} \alpha_j R_j(\bar{\eta})$. In order to minimize the total cost, we have to find a schedule minimizing the sum of the weighted number of tardy jobs and the batch-delivery costs with respect to $d_j(\bar{\eta})$, denoted by $1|s, d_j(\bar{\eta})|\sum w_j U_j + bq$. This is actually a $1|s|\sum w_j U_j + bq$ problem in which $d_j(\bar{\eta})$ is the given due date of job j, $\forall j \in J$.

Before we describe our Algorithm CH8-A3 which finds an optimal solution, we need to determine an upper bound for $\bar{\eta}$. Since there are at most n batches in any

feasible schedule, the largest possible makespan is $P + ns$. It means that the largest possible value for any acting due date is $P+ns$, i.e., $d_j(\bar{\eta}) = \max\{\bar{\eta}p_j, A_j\} \leq P+ns$, $\forall j \in J$. Let the set χ_j contain all possible values for η such that $D_j(\eta) = \eta p_j \in [0, P + ns]$ and $D_j(\eta)$ is integer, $\forall \eta \in \chi_j$. Thus we have

$$\chi_j = \{0, \frac{1}{p_j}, \frac{2}{p_j}, ..., \frac{P+ns}{p_j}\}, \forall j \in J. \tag{8.3.18}$$

Let $\mathcal{X} = \chi_1 \cup \chi_2 \cup ... \cup \chi_n$ be the set which stores all possible values for $\bar{\eta}$. For some $\bar{\eta}$, it may happen that $D_j(\bar{\eta}) = \bar{\eta}p_j$ is not an integer. In this situation, we will take the value of $D_j(\bar{\eta}) = \lfloor \bar{\eta}p_j \rfloor$. We now describe an algorithm to solve this problem.

Algorithm CH8-A3

[Initialization]: Set $v^* = \infty$ and determine $\chi_1, ..., \chi_n$ and $\mathcal{X} = \chi_1 \cup ... \cup \chi_n$.

For each $\bar{\eta} \in \mathcal{X}$

1. Determine $d_j(\bar{\eta}) = \max\{D_j(\bar{\eta}), A_j\}$ and $R_j(\bar{\eta}) = \max\{D_j(\bar{\eta}) - A_j, 0\}$, where $D_j(\bar{\eta}) = \lfloor \bar{\eta}p_j \rfloor$, $\forall j \in J$.
2. Run Algorithm CH4-A1 for the $1|s, d_j(\bar{\eta})|\sum w_jU_j + bq$ problem and update $v^* = \min\{\sum_{j=1}^{n} \alpha_j R_j(\bar{\eta}) + v(\bar{\eta}), v^*\}$. */*$v(\bar{\eta})$ is the optimal solution value obtained by Algorithm CH4-A1 for the $1|s, d_j(\bar{\eta})|\sum w_jU_j + bq$ problem.*

Endfor

[Result]: Trace back v^*, $\bar{\eta}$ and $v(\bar{\eta})$ to obtain the optimal schedule.

Theorem 8.3.2 *Algorithm CH8-A3 is a pseudo-polynomial algorithm, which finds an optimal solution for the TWK problem in $O(n^4[\min\{P + ns, W + nq\}][P + ns])$ time and space, where $P = \sum_{j=1}^{n} p_j$ and $W = \sum_{j=1}^{n} w_j$. This shows that the TWK problem is $\mathcal{NP}$-hard only in the ordinary sense.*

Proof. We know that Algorithm CH4-A1 runs in $O(n^3[\min\{d_{\max}(\bar{\eta}), P+ns, W+nq\}])$ time and space for the $1|s, d_j(\bar{\eta})|\sum w_jU_j + bq$ problem by Theorem 4.3.1. Since the

largest possible acting due date is satisfying $d_{\max}(\bar{\eta}) = \bar{\eta}p_{\max} = \frac{P+ns}{p_{\min}}p_{\max} \geq P + ns$, where $p_{\max} = \max_{j\in J}\{p_j\}$, we have $\min\{d_{\max}(\bar{\eta}), P+ns, W+nq\} = \min\{P+ns, W+nq\}$. Moreover, Algorithm CH8-A3 calls Algorithm CH4-A1 at most $|X| = \sum_{j=1}^{n}|\chi_j|$ times, where $|\chi_j| = P+ns+1$ by equation (8.3.18). Thus, the complexity of Algorithm CH8-A3 is $O(n^4[\min\{P+ns, W+nq\}][P+ns])$. ■

8.3.3 Fully Polynomial Time Approximation Scheme

Once again, in an approach analogous to that for the SLK problem, suppose that there are g distinct $\frac{A_j}{p_j}$ values, i.e.,

$$1 < \frac{A_{\{1\}}}{p_{\{1\}}} < ... < \frac{A_{\{g\}}}{p_{\{g\}}} < \frac{P+ns}{p_{\min}}. \tag{8.3.19}$$

The due-date-assignment cost for $\eta = x$ can be represented as

$$\sum_{j=1}^{n}\alpha_j R_j(x) = \sum_{j=1}^{g}\hat{\alpha}_{\{j\}}R_{\{j\}}(x) = \sum_{j=1}^{g}\hat{\alpha}_{\{j\}}\max\{x - \frac{A_{\{j\}}}{p_{\{j\}}}, 0\}, \tag{8.3.20}$$

where $\hat{\alpha}_{\{j\}} = \sum_{i\in\hat{J}_{\{j\}}}\alpha_i p_i$ and $\hat{J}_{\{j\}} = \{i|A_i/p_i = A_{\{j\}}/p_{\{j\}}, \forall i \in J\}$. This function is also a piecewise linear function of x and is also denoted by bold and sloped lines in Figure 8.2. When x passes a break point in equation (8.3.19), $\sum_{j=1}^{g}\hat{\alpha}_{\{j\}}R_{\{j\}}(x)$ will include one more job set, say job $\hat{J}_{\{j\}}$, with a positive cost and the slope will increase by $\hat{\alpha}_{\{j\}}$. Let us further define $\eta_0 = 1$, $\eta_i = \frac{A_{\{i\}}}{p_{\{i\}}}$, $i = 1, ..., g$ and $\eta_{g+1} = \frac{P+ns}{p_{\min}}$. For each interval $[\eta_i, \eta_{i+1})$, $i = 1, ..., g$, partition it into m_i smaller intervals:

$$\begin{aligned}
&H_{i,1}^{TWK} = (\eta_i, \eta_i + \hat{\delta}_i];\\
&H_{i,2}^{TWK} = (\eta_i + \hat{\delta}_i, \eta_i + \hat{\delta}_i + \hat{\delta}_i(1+\varepsilon)];\\
&\qquad .....\\
&H_{i,m_i}^{TWK} = (\eta_i + \hat{\delta}_i\sum_{k=0}^{m_i-2}(1+\varepsilon)^k, \min\{\eta_{i+1}, \eta_i + \hat{\delta}_i\sum_{k=0}^{m_i-1}(1+\varepsilon)^k\}],
\end{aligned} \tag{8.3.21}$$

where m_i is such that

$$\eta_i + \hat{\delta}_i \sum_{k=0}^{m_i-2} (1+\varepsilon)^k \leq \eta_{i+1} \leq \eta_i + \hat{\delta}_i \sum_{k=0}^{m_i-1} (1+\varepsilon)^k$$

and thus

$$\hat{\delta}_i = 1 + \varepsilon + \varepsilon \frac{\sum_{j=1}^{i} \hat{\alpha}_{\{j\}}(\eta_i - \eta_j)}{\sum_{j=1}^{i} \hat{\alpha}_{\{j\}}}. \tag{8.3.22}$$

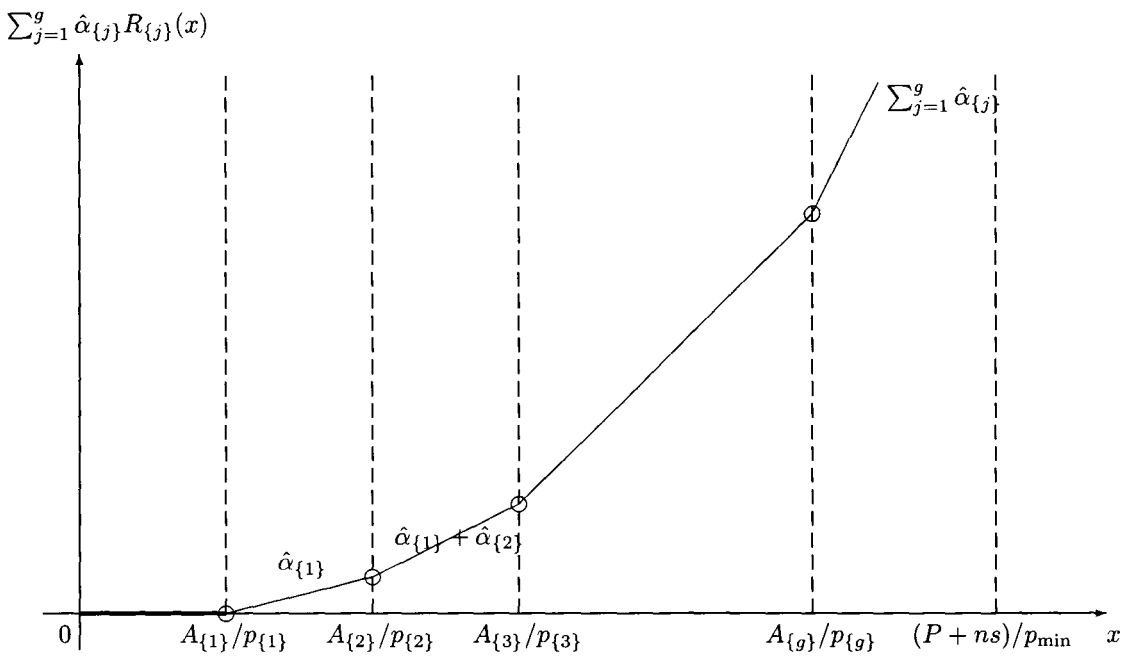


Figure 8.2: The TWK due-date-assignment cost as a piecewise linear function of x.

It is sufficient to chose m_i to be

$$m_i = \lceil \log_{(1+\varepsilon)} \frac{\varepsilon(\eta_{i+1} - \eta_i)}{\hat{\delta}_i} + 1 \rceil. \tag{8.3.23}$$

We note that

$$O(m_i) \leq O(\log \eta_{i+1}) \leq O(\log(P + ns)). \tag{8.3.24}$$

Let $\sigma(\tilde{\eta})$ be an optimal schedule for the TWK problem and let $\eta = \tilde{\eta}$ be the corresponding coefficient value. Then the total cost is

$$v^*(\tilde{\eta}) = \sum_{j=1}^{n} \alpha_j R_j(\tilde{\eta}) + \sum_{j=1}^{n} w_j U_j(\sigma(\tilde{\eta})) + b(\sigma(\tilde{\eta}))q. \tag{8.3.25}$$

It is true that schedule $\sigma(\tilde{\eta})$ is also an optimal schedule for the $1|s, d_j(\tilde{\eta})|\sum w_j U_j(\tilde{\eta}) + bq$ problem. Suppose $\tilde{\eta} \in H_{i,k}^{TWK}$ for some $0 < i \leq g$ and $0 < k \leq m_i$. Let

$$\eta' = \begin{cases} \eta_i + \hat{\delta}_i \sum_{r=0}^{k-1}(1+\varepsilon)^r, & \text{if } 0 < k < m_i \\ \min\{\eta_{i+1}, \eta_i + \hat{\delta}_i \sum_{r=0}^{k-1}(1+\varepsilon)^r\}, & \text{if } k = m_i \end{cases} \tag{8.3.26}$$

be the right end point of the interval $H_{i,k}^{TWK}$. Since $\tilde{\eta} \in H_{i,k}^{TWK}$, then we have $\eta' \geq \tilde{\eta}$ and thus

$$d_j(\eta') = \max\{\eta' p_j, A_j\} \geq d_j(\tilde{\eta}) = \max\{\tilde{\eta} p_j, A_j\}, \forall j \in J. \tag{8.3.27}$$

Similarly to equations (8.2.13) and (8.2.14), for $k = 1$, we have

$$\begin{aligned} \sum_{j=1}^{n} \alpha_j R_j(\eta') &= \sum_{j=1}^{h} \hat{\alpha}_{\{j\}} \max\{\eta' - \frac{A_{\{j\}}}{p_{\{j\}}}, 0\} \\ &\leq \sum_{j=1}^{i} \hat{\alpha}_{\{j\}}[\eta_i + \hat{\delta}_i - \eta_j] \\ &= \sum_{j=1}^{i} \hat{\alpha}_{\{j\}}[\eta_i + 1 + \varepsilon + \varepsilon \frac{\sum_{j=1}^{i} \hat{\alpha}_{\{j\}}(\eta_i - \eta_j)}{\sum_{j=1}^{i} \hat{\alpha}_{\{j\}}} - \eta_j] \\ &= (1+\varepsilon) \sum_{j=1}^{i} \hat{\alpha}_{\{j\}}(\eta_i + 1 - \eta_j) \\ &\leq (1+\varepsilon) \sum_{j=1}^{i} \hat{\alpha}_{\{j\}}(\tilde{\eta} - \eta_j) \\ &= (1+\varepsilon) \sum_{j=1}^{n} \alpha_j R_j(\tilde{\eta}), \end{aligned} \tag{8.3.28}$$

and for $k > 1$, we have

$$\begin{aligned}
\sum_{j=1}^{n} \alpha_j R_j(\eta') &= \sum_{j=1}^{h} \hat{\alpha}_{\{j\}} \max\{\eta' - \frac{A_{\{j\}}}{p_{\{j\}}}, 0\} \\
&\leq \sum_{j=1}^{i} \hat{\alpha}_{\{j\}} [\eta_i + \hat{\delta}_i \sum_{r=0}^{k-1} (1+\varepsilon)^r - \eta_j] \\
&= \sum_{j=1}^{i} \hat{\alpha}_{\{j\}} (\eta_i - \eta_j) + \sum_{j=1}^{i} \hat{\alpha}_{\{j\}} [\hat{\delta}_i \sum_{r=1}^{k-1} (1+\varepsilon)^r] + \sum_{j=1}^{i} \hat{\alpha}_{\{j\}} [\hat{\delta}_i (1+\varepsilon)^0] \\
&= (1+\varepsilon) \sum_{j=1}^{i} \hat{\alpha}_{\{j\}} (\eta_i - \eta_j) + \sum_{j=1}^{i} \hat{\alpha}_{\{j\}} [\hat{\delta}_i \sum_{r=1}^{k-1} (1+\varepsilon)^r] + (1+\varepsilon) \sum_{j=1}^{i} \hat{\alpha}_{\{j\}} \\
&= (1+\varepsilon) \sum_{j=1}^{i} \hat{\alpha}_{\{j\}} [\eta_i + \hat{\delta}_i \sum_{r=0}^{k-2} (1+\varepsilon)^r + 1 - \eta_j] \\
&\leq (1+\varepsilon) \sum_{j=1}^{i} \hat{\alpha}_{\{j\}} (\tilde{\eta} - \eta_j) \\
&= (1+\varepsilon) \sum_{j=1}^{n} \alpha_j R_j(\tilde{\eta}), \qquad (8.3.29)
\end{aligned}$$

where $R_j(\eta') = \max\{\eta' p_j - A_j, 0\}$ and $R_j(\tilde{\eta}) = \max\{\tilde{\eta} p_j - A_j, 0\}$, $\forall j \in J$. In the last term of the third line, we substitute $\hat{\delta}_i$ by equation (8.3.22), and the last inequality holds because $\eta_i + \hat{\delta}_i \sum_{r=0}^{k-2} (1+\varepsilon)^r < \tilde{\eta}$.

Let $w_{opt}(\eta')$ be the optimal solution value for the $1|s, d_j(\eta')| \sum w_j U_j(\eta') + bq$ problem. By equation (8.3.27), we know that

$$w_{opt}(\eta') \leq \sum_{j=1}^{n} w_j U_j(\sigma(\tilde{\eta})) + b(\sigma(\tilde{\eta}))q. \qquad (8.3.30)$$

Let $\sigma_{apx}(\eta')$ be a schedule for the $1|s, d_j(\eta')| \sum w_j U_j(\eta') + bq$ problem with cost

$$w_{apx}(\eta') \leq (1+\varepsilon) w_{opt}(\eta'). \qquad (8.3.31)$$

Then the total cost of the TWK problem on schedule $\sigma_{apx}(\eta')$ is

$$\begin{aligned} v(\eta') &= \sum_{j=1}^{n} \alpha_j R_j(\eta') + w_{apx}(\eta') \\ &\leq (1+\varepsilon) \sum_{j=1}^{n} \alpha_j R_j(\tilde{\eta}) + (1+\varepsilon) w_{opt}(\eta') \\ &\leq (1+\varepsilon) \sum_{j=1}^{n} \alpha_j R_j(\tilde{\eta}) + (1+\varepsilon)[\sum_{j=1}^{n} w_j U_j(\sigma(\tilde{\eta})) + b(\sigma(\tilde{\eta}))q] \\ &\leq (1+\varepsilon) v^*(\tilde{\eta}), \end{aligned} \tag{8.3.32}$$

because of the previous equations (8.3.28), (8.3.29), (8.3.30) and (8.3.31).

Similarly to the FPTAS for the SLK problem, Algorithm CH8-A4 calls Algorithm CH4-A7 for each $1|s, d_j(\eta')| \sum w_j U_j(\eta') + bq$ problem for each possible η' in equation (8.3.26). For some η', it may happen that some $D_j(\eta') = p_j \eta'$ is not an integer. In this situation, we will take the value of $D_j(\eta') = \lfloor p_j \eta' \rfloor$.

Algorithm CH8-A4

[Initialization]: Set $\mathcal{T} = \emptyset$ and determine g, ξ_i, η_i and m_i for any given $\varepsilon > 0$ based on the above discussion.

For $i = 1$ **to** g

For $k = 1$ **to** m_i

1. Set η' by equation (8.3.26).
2. Determine $d_j(\eta') = \max\{D_j(\eta'), A_j\}$ and $R_j(\eta') = \max\{D_j(\eta') - A_j, 0\}$, where $D_j(\eta') = \lfloor \eta' p_j \rfloor$, $\forall j \in J$.
3. Run Algorithm CH4-A7 for the $1|s, d_j(\eta')| \sum w_j U_j + bq$ problem and obtain $v(\eta')$. */*$v(\eta')$ is the approximate solution value obtained by Algorithm CH4-A7 for the $1|s, d_j(\eta')| \sum w_j U_j + bq$ problem.*
4. Set $\mathcal{T} \leftarrow \mathcal{T} \cup (\eta', \sum_{j=1}^{n} \alpha_j R_j(\eta') + v(\eta'))$.

Endfor

Endfor

[Result]: Select the state with the smallest $\sum_{j=1}^{n} \alpha_j R_j(\eta') + v(\eta')$ in $\mathcal{T}$ and trace back to obtain the schedule.

Theorem 8.3.3 *Algorithm CH8-A4 is an FPTAS, which finds a* $(1+\varepsilon)$*-approximate solution to the TWK problem in* $O(\sum_{i=1}^{g} m_i[n^4 \log\log n + n^4/\varepsilon])$ *time for any given* $\varepsilon > 0$, *where* m_i *is defined in equation (8.3.23).*

Proof. The correctness of the algorithm follows from the preceding discussion and Theorem 4.4.4. Algorithm CH4-A7 runs in $O(n^4 \log\log n + n^4/\varepsilon)$ time and Algorithm CH8-A4 calls Algorithm CH4-A7 at most $O(\sum_{i=1}^{g} m_i)$ times. Note that $O(m_i) \leq O(\log(P + ns))$. Therefore, Algorithm CH8-A4 runs in $O(\sum_{i=1}^{g} m_i[n^4 \log\log n + n^4/\varepsilon]) \leq O(\log(P + ns)[n^5 \log\log n + n^5/\varepsilon])$ time. ■

8.4 Summary

In this chapter, we studied two supply chain scheduling problems with delivery costs and due date assignments: the SLK and the TWK problem. For each problem, we proved its $\mathcal{NP}$-hardness and found an pseudo-polynomial algorithm and an FPTAS.

Chapter 9

Conclusions and Future Research

In this thesis, we studied supply chain scheduling problems *only* with delivery costs (Chapters 4 and 5) and supply chain scheduling problems with *both* delivery costs *and* due date assignment (Chapters 6, 7 and 8). For the above problems, we studied their computational complexity, designed and analyzed efficient algorithms for their optimal solutions and for their approximate solutions.

In Chapter 3, we first introduced terminologies, assumptions and notations which were used in later Chapters. Then we proposed an algorithm framework, the bound improvement procedure and proved its complexity and correctness. This procedure was used in Chapters 4, 5, 6 and 7 to narrow a pair of initial bounds into a pair of tight bounds which were used in developing approximation algorithms.

In Chapter 4, we studied the single-customer problem where our goal is to minimize the sum of the weighted number of tardy jobs and batch-delivery costs on a single machine. Since it is $\mathcal{NP}$-hard, we proposed a pseudo-polynomial algorithm, which further demonstrated that the problem is $\mathcal{NP}$-hard only in the ordinary sense. The pseudo-polynomial algorithm finds optimal solutions in polynomial time for two special cases: one case has equal processing times and one case has equal tardiness penalties. Finally, we converted the pseudo-polynomial algorithm into an FPTAS for the general case.

In Chapter 5, we studied the multiple-customer problem where our goal is to minimize the sum of the weighted number of tardy jobs and batch-delivery costs on a single machine. We first provided a $\mathcal{NP}$-hardness proof for it. Then we proposed an exact algorithm which runs in pseudo-polynomial time. This implies that the problem is $\mathcal{NP}$-hard only in the ordinary sense. In order to develop an approximation algorithm, we studied the problem with the restriction that tardy jobs require separate deliveries. With this restriction, we were able to present an FPTAS for it. Finally, this FPTAS was proven to be a good approximation algorithm for the original problem.

In Chapter 6, we studied the CON problems, where our goal is to minimize the sum of the weighted number of tardy jobs, the due-date-assignment costs and the batch-delivery costs on a single machine for a single customer and CON means that a common due date is assigned to all jobs. We defined and studied three problems: the unconstrained problem, the time-constrained problem and the capacity-constrained problem. For these three problems, we were able to provide $\mathcal{NP}$-hardness proofs, pseudo-polynomial algorithms and FPTAS, respectively.

In Chapter 7, we studied the DIF problems, where our goal is to minimize the sum of the weighted number of tardy jobs, the due-date-assignment costs and the batch-delivery costs on a single machine for a single customer and DIF means that distinct due dates can be assigned to each job individually. We first provided a strong $\mathcal{NP}$-hardness proof for the problem with arbitrary due-date-assignment costs. Then we changed our focus to the problem with equal due-date-assignment costs. For this problem, we proved that it is $\mathcal{NP}$-hard and found a pseudo-polynomial algorithm, which further classified that the problem is $\mathcal{NP}$-hard only in the ordinary sense. Then we converted the pseudo-polynomial algorithm into an FPTAS for the problem with equal due-date-assignment costs. Finally, we found a polynomial algorithm for the problem which has equal due-date-assignment costs and equal tardiness penalties.

In Chapter 8, we studied the SLK problem and the TWK problem, where our goal is to minimize the sum of the weighted number of tardy jobs, the due-date-assignment costs and the batch-delivery costs on a single machine for a single

customer. In the SLK problem, the assigned due dates are determined by adding a common slack to the processing times and in the TWK problem, the assigned due dates are determined by multiplying a common coefficient to the processing times. For both problems, we first proved that they are $\mathcal{NP}$-hard. Then, by implementing the pseudo-polynomial algorithm in Chapter 4, we developed pseudo-polynomial algorithms, which established that both problems are $\mathcal{NP}$-hard only in the ordinary sense. Finally, by implementing the FPTAS in Chapter 4, we proposed FPTAS for them, respectively.

Further research in this area may look at other scheduling objectives such as the maximum tardiness and the total tardiness in problems with both delivery costs and due date assignment. Another possible direction may change efforts to the type of deliveries including routing. In this thesis, we proved the strong $\mathcal{NP}$-hardness of several problems. The design and analysis of good heuristics and approximation algorithms for these problems may be the subject of further research.

In all the models studied in this thesis, we have job processing times as predetermined and unchangeable parameters. Over the past several decades, however, a large number of suppliers intended to coordinate with outsourcing partners. If a supplier wants to finish an order on time with its limited production capacity, in most cases, the supplier may want to outsource part of the order to a third party rather than expand its own capacity. Therefore, the supplier needs to produce only the remaining (or not outsourced) part of the order. This can be modeled by reduced processing times, also called controllable processing times. We can see a large number of scheduling literature dealing with this type of research, and for the most recent development we refer to the survey paper [Shabtay and Steiner, 2007]. By including controllable processing times into the models studied in Chapters 6, 7 and 8, we can open up a new research area, *supply chain scheduling with due date and outsourcing coordination*, where suppliers coordinate not only with customers but also with their partners.

Bibliography

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

M. Akker and H. Hoogeveen. *Minimizing the Number of Tardy Jobs.* Springer-Verlag, 2004.

K.R. Baker and G.D. Scudder. Sequencing with earliness and tardiness penalties: A review. *Operations Research*, 38:22–36, 1990.

P. Baptiste. An $o(n^4)$ algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Operations Reserch Letters*, 24:175–180, 1999.

A. Bar-Noy, R. Bar-Yehuda, A. Freund, J Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5): 1060–1090, 2001.

P. Brucker. *Scheduling Algorithms.* Springer-Verlag New York, Inc., 3rd edition, 2001.

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

B. Chen, C.N. Potts, and G.J. Woeginger. 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, 1998.

Z.-L. Chen. Integrated production and outbound distribution scheduling: Review and extension. *To appear in Operations Research*, 2008.

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

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

Z.-L. Chen and G.L. Vairaktarakis. Integrated scheduling of production and distribution operations. *Management Science*, 51:614–628, 2005.

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

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

S. Chudanov, M. Kovalyov, and E. Pesch. An FPTAS for a single-item capacity economic lot-sizing problem with monotone cost structure. *Mathematical Programming, Series A*, 106:453–466, 2006.

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

W.J. Cook, W.R. Cunningham, W.R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization.* J. Wiley, 1998.

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

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

C.V. Gens and E.V. Levner. Computational complexity of approximation algorithms for combinatorial problems. *Lecture Notes in Computer Science*, 74:292–300, 1979a.

G.V. Gens and E.V. Levner. Discrete optimization problems and efficient approximate algorithms. *Engineering Cybernetics*, 17(6):1–11, 1979b.

G.V. Gens and E.V. Levner. Fast approximation algorithm for job sequencing with deadlines. *Discrete Applied Mathematics*, 3(4):313–318, 1981.

V.S. Gordon, J.-M. Proth, and C. Chu. A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal of Operational Research*, 139(3):1–25, 2002a.

V.S. Gordon, J.-M. Proth, and C. Chu. Due date assignment and scheduling: SLK, TWK and other due date assignment models. *Production Planning and Control*, 13(2):117–132, 2002b.

V.S. Gordon, J.-M. Proth, and V.A. Strusevich. Scheduling with due date assignment. *In Handbook of Scheduling: Algorithms, Models and Performance Analysis, Edited by Joseph Y-T. Leung*, 21:1–22, 2004.

R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 4:287–326, 1979.

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

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

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

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

D.S. Hochbaum and D. Landy. Scheduling with batching: Minimizing the weighted number of tardy jobs. *Operations Research Letters*, 16:79–86, 1994.

P. Kaminsky and D.S. Hochbaum. Due date quotation models and algorithms. *In Handbook of Scheduling Algorithms, Models and Performance Analysis, Edited by Joseph Y-T. Leung*, 20:1–22, 2004.

R.M. Karp. Reducibility among combinatorial problems. *In Complexity of Computer Computations, Edited by R.E. Miller and J.W. Thatcher*, pages 85–103, 1972.

E.L. Lawler. Scheduling a single machine to minimize the number of late jobs. *Report No. USB/CSD/83/139, Computer Science Division, University of California, Berkeley, USA*, 1983.

E.L. Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Annals of Operations Research*, 26: 125–133, 1990.

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

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

J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Operations Research*, 1:343–362, 1977.

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

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

J.M. Moore. An n job, one machine sequencing algorithm for minimzing the number of late jobs. *Management Science*, 15:102–109, 1968.

C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, MA, 1994.

M. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Prentice-Hall, 2nd edition, 2001.

C.N. Potts and Y.M. Kovalyov. Scheduling with batching: A review. *European Journal of Operational Research*, 120:228–249, 2000.

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

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

S.K. Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM*, 23 (1):116–127, 1976.

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

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

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

D. Shabtay and G. Steiner. Two due date assignment problems in scheduling a single machine. *Operations Research Letters*, 34(6):683–691, 2006.

D. Shabtay and G. Steiner. A survey of scheduling with controllable processing times. *Discrete Applied Mathematics*, 155(13):1643–1666, 2007.

S.A. Slotnick and M.J. Sobel. Manufacturing lead-time rules: Customer retention versus tardiness costs. *European Journal of Operational Research*, 169:825–856, 2005.

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

D.J. Thomas and P.M. Griffin. Coordinated supply chain management. *European Journal of Operational Research*, 94:1–15, 1996.

V.V. Vazirani. *Approximation Algorithms*. Springer, Berlin, German, 2003.

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

G.J. Woeginger. Open problems around exact algorithms. *Discrete Applied Mathematics*, 156:397–405, 2008.

X.G. Yang. Scheduling with generalized batch delivery dates and earliness penalties. *IIE Transactions*, 32:735–741, 2000.

Appendix A

Terminologies

- *Batch-completion time:* the completion time of the last job in a batch;
- *Batch-due date:* the smallest due date of the jobs in a batch;
- *Early batch:* a batch includes only early jobs;
- *Tardy batch:* a batch includes only tardy jobs;
- *Mix batch:* a batch includes both early and tardy jobs.

Appendix B

Assumptions

- *Unlimited-delivery:* the number of vehicles available for delivering jobs is infinity;
- *Instant-delivery:* it takes time zero to deliver a batch to any customers;
- *Tardy-delivery:* all jobs have to be delivered to customers including tardy jobs;
- *Batch-delivery:* only jobs for the same customer can be delivered in a batch;
- *Single-machine:* all inbound operations are modeled as a single machine;
- *Non-preemption:* interruption is not allowed during the processing of a job;
- *Zero-availability:* all jobs are available for processing at time zero;
- *Non-negativity:* all data are non-negative integers;
- *Due-date-adjustment:* all due dates have been adjusted by subtracting an associated delivery time.

Appendix C

Notations

- $J = \{1, ...n\}$: the set of jobs;
- p_j : the processing time of job j;
- d_j : the due date of job j;
- 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;
- D : the common assigned due date in the CON problems, i.e., $D_j = D, \forall j \in J$, which is a decision variable;
- 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;
- α_j : the due-date-assignment cost per extended time unit of R_j;
- α : the uniform due-date-assignment cost per extended time unit of R_j, i.e., $\alpha_j = \alpha, \forall j \in J$;

- $\alpha_j R_j$: 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;
- s : the batch-setup time before processing the first job in each batch;
- τ : the batch-delivery time of each batch;
- q : the batch-delivery cost of each batch;
- b : the number of batches in a schedule;
- T : the minimum time between any two consecutive deliveries;
- B : the maximum number of jobs in a single batch (delivery);
- θ : a slack variable in the SLK problem, where $D_j(\theta) = \theta + p_j$, $\forall j \in J$;
- $d_j(\theta) = \max\{D_j(\theta), A_j\}$ is the acting due date in the SLK problem;
- η : a coefficient variable in the TWK problem, where $D_j(\eta) = \eta p_j$, $\forall j \in J$;
- $d_j(\eta) = \max\{D_j(\eta), A_j\}$ is the acting due date in the TWK problem;
- $M = \{1, ..., m\}$: the set of customers;
- n_i : the number of jobs for customer i and $n = \sum_{i=1}^{m} n_i$;
- $J_i = \{(i, 1), ..., (i, n_i)\}$: the set of jobs for customer i and $J = \{J_1, ..., J_m\}$;
- $p_{(i,k)}$: the processing time of job (i, k);

- $d_{(i,k)}$: the due date of job (i,k);
- $w_{(i,k)}$: the tardiness penalty (weight) of job (i,k);
- $C_{(i,k)}$: the batch-completion time of job (i,k);
- $U_{(i,k)}$: the tardiness indicator: $U_{(i,k)} = 1$, if job (i,k) is tardy and $U_{(i,k)} = 0$ otherwise;
- s_i : the batch-setup time before the first job in each batch of customer i;
- τ_i : the batch-delivery time of each batch for customer i;
- q_i : the batch-delivery cost of each batch for customer i;
- b_i : the number of batches for customer i in a schedule;
- SEP implies that the tardy jobs need to be delivered separately.

Appendix D

Models and Results

Model	Complexity	Theorem
$1\|s\|\sum w_jU_j+bq$	$O(n^4\log\log n+n^4/\varepsilon)$	4.4.4
$1\|s_i\|\sum w_{(i,k)}U_{(i,k)}+b_iq_i$	$\mathcal{NP}$-hard in the ordinary sense	5.4.1
$1\|s_i,SEP\|\sum w_{(i,k)}U_{(i,k)}+b_iq_i$	$O(n^{m+2}\log\log n+n^{m+2}/\varepsilon)$	5.4.5
$1\|s,A>0,CON\|\sum\alpha_jR_j+\sum w_jU_j+bq$	$O(n^2/\varepsilon)$	6.3.3
$1\|s,T,A>0,CON\|\sum\alpha_jR_j+\sum w_jU_j+bq$	$O(n^2/\varepsilon)$	6.4.3
$1\|s,B,A,CON\|\sum\alpha_jR_j+\sum w_jU_j+bq$	$O(l[n^2\log\log n+n^2/\varepsilon])$	6.5.5
$1\|s,B,A,d_j=x\|\sum w_jU_j+bq$	$O(n^2\log\log n+n^2/\varepsilon)$	6.5.4
$1\|s,A,DIF\|\sum\alpha_jR_j+\sum w_jU_j+bq$	Strongly $\mathcal{NP}$-hard	7.2.1
$1\|s,A,DIF\|\sum\alpha R_j+\sum w_jU_j+bq$	$O(n^5/\varepsilon+n^5\log\log n)$	7.3.5
$1\|s,A_j,SLK\|\sum\alpha_jR_j(\theta)+\sum w_jU_j(\theta)+bq$	$O(\sum_{i=1}^{h}l_i[n^4\log\log n+n^4/\varepsilon])$	8.2.3
$1\|s,A_j,TWK\|\sum\alpha_jR_j(\eta)+\sum w_jU_j(\eta)+bq$	$O(\sum_{i=1}^{g}m_i[n^4\log\log n+n^4/\varepsilon])$	8.3.3