Single Machine Scheduling with Release Dates and Due Dates

Zhongjun Tian

Ph.D.

THE HONG KONG POLYTECHNIC UNIVERSITY

2003

Single Machine Scheduling with Release Dates and Due Dates

by

Zhongjun Tian

A thesis submitted to
The Hong Kong Polytechnic University
for the degree of
Doctor of Philosophy

Under the Supervisions of
Dr C.T. Daniel Ng
Professor T.C. Edwin Cheng

Department of Management
The Hong Kong Polytechnic University
2003

Abstract of the thesis entitled **'Single Machine Scheduling with Release Dates and Due Dates'**

submitted by Zhongjun Tian

for the degree of Doctor of Philosophy

at The Hong Kong Polytechnic University in June 2002

In this thesis, we study some classical single machine scheduling problems with release dates and due dates. In a comprehensive review of prior works, we classify the literature into different classes, according to the job characteristics and the optimality criteria. The review reveals that, despite the particular importance of single machine scheduling models, a few classes of the problems on this topic are rarely or never touched. These classes are worthy of research because of the universality of the scheduling models with non-simultaneously released jobs and the practical significance of due-date-based research. This thesis focuses on two of these classes, which are the preemptive scheduling on a single machine to minimize total tardiness with job restrictions, and the single machine due date assignment with release dates.

In the former class, a set of jobs has to be processed on a single machine that can perform only one job at a time. Each job has a release date, a processing time, and a due date. All the data are integers. Preemption is allowed. The objective is to schedule the jobs so as to minimize the total tardiness. We study two special cases of this $\mathcal{NP}$-hard problem with the following restrictions, separately:

(1) All processing times are equal, while the release dates and due dates are arbitrary.

(2) The processing times are arbitrary, while the release dates and due dates are agreeable, namely, all release dates and due dates are similarly ordered.

For the second case, we consider two subcases where identical release dates correspond to identical and arbitrary due dates, respectively. For each of the specified problems, we investigate the optimality properties and develop an appropriate algorithm. Some of the results are extended to the single machine scheduling problem without release dates.

In the latter class, a set of jobs has to be processed on a single machine that can process no more than one job at a time. Each job has a release date, a processing time, a due date and a weight. All the data are integers. The processing times and release dates are arbitrary, while the weights are either arbitrary or identical. The due dates are assigned by three different methods:

(1) Constant (CON): all jobs are given exactly the same flow allowance.

(2) Slack (SLK): jobs are given flow allowances that reflect equal slacks.

(3) Total-work-content (TWK): due dates are based on total work content.

The objective is to schedule the jobs so as to minimize one of the following criteria: the maximum tardiness, the (weighted) number of tardy jobs and the total (weighted) tardiness. We examine the complexity of all the problems with the consideration of the due date assignment methods and the optimality criteria. For each of the *NP*-hard problems, we provide an *NP*-hardness proof; and for each of the solvable problems, we introduce a polynomial or pseudo-polynomial algorithm.

Acknowledgements

It is my great pleasure to thank all those who have supported and contributed to the completion of this Ph.D. thesis.

The great enthusiasm, encouragement and efficient guidance that I have received from Dr C.T. Daniel Ng, my chief-supervisor, and Professor T.C. Edwin Cheng, my co-supervisor, have been beneficial not only for the completion of the thesis but also for the long-term development of my research career.

This Ph.D. study is financially supported by The Hong Kong Polytechnic University grant number G-V625. The support and financial aid from this university are greatly appreciated.

I sincerely thank all of my colleagues and office-mates at The Hong Kong Polytechnic University. I am very pleased to take this opportunity to thank my wife and my son. Without their understanding and support, I can never make this accomplishment of my Ph.D. study come to be true.

Zhongjun Tian
Hong Kong
June 2002

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Single machine models are important in machine scheduling systems. On the one hand. single machine environment is a particular case of many industrial and services environments. On the other hand, scheduling problems with more complicated environments can often be decomposed into subproblems that deal with a single machine. For example, a complicated environment with a single bottleneck may give rise to a single machine model. So, the results that can be obtained for single machine not only provide insights into the single machine environment, but also provide a basis for heuristics for more complicated environments (Pinedo 1995). Thereby, single machine scheduling problems have been studied by a great number of researchers since the seminal work by Jackson (1955) and Smith (1956), and an impressive amount of literature has been created.

Although the single machine environment seems simple, the corresponding scheduling problems could be rather difficult in case with complicated job characteristics, even for usual optimality criteria. Especially, when jobs are not released simultaneously, single machine scheduling problems for minimizing all common due-date-based criteria are *NP*-hard. As early as 25 years ago, Lenstra *et al.* (1977) proved the strong *NP*-hardness of all these problems, i.e. the single machine scheduling problems with release dates to minimize the maximum lateness, the (weighted) number of tardy jobs and the total (weighted) tardiness, respectively. However, because of the universality of the scheduling models with non-simultaneously released jobs, researchers never lose interest in these *NP*-hard problems in the past decades. The study on these problems is concentrated on two areas: the enumerative and approximation algorithms for the general problems, and the polynomial solutions for the special cases with job restrictions. In both areas, considerable amount of papers with attractive results have been published. Nevertheless, a comprehensive review shows that there are still a few sections of both areas rarely or never touched. In this thesis, we focuses on one of such sections in the latter area, which is the preemptive single machine total tardiness problem with job restrictions. Besides, we examine and tackle some other single machine problems, which are the single machine due date assignment problems with release dates.

1.1 Preemptive total tardiness problem

An up to date classification for complexity results of scheduling problems can be found on a web site maintained by Brucker and Knust (2002). Our general observation from the classification is that, in most cases, preemption can make an *NP*-hard single machine scheduling problem easier. If job weights are not considered, all the strongly *NP*-hard problems with release dates become polynomially solvable except the total tardiness problem. Firstly, the preemptive single machine total completion time problem with release dates can be solved by the well-known shortest remaining processing time (SRPT) rule. Secondly, Baker *et al.* (1983) and Gordon and Tanaev (1983) independently presented an $O(n^2)$ algorithm which solves the preemptive single machine maximum lateness problem with release dates and arbitrary precedence constraints, where n is the number of the jobs. Lastly, Lawler (1990) provided an $O(n^5)$ algorithm for the preemptive single machine number of tardy jobs problem with release dates, which can also be solved by an $O(n^4)$ algorithm proposed by Baptiste (1999a).

On the other hand, according to Chu (1992), the preemptive single machine total tardiness problem with release dates is *NP*-hard, since the counterpart without release dates is *NP*-hard and the former is not easier than the latter. But, as the latter problem is *NP*-hard in the ordinary sense, the sense of the *NP*-hardness of the former problem cannot be determined without a pseudo-polynomial algorithm or a reduction from a strongly *NP*-hard problem. So, the preemptive single machine total tardiness problem with release dates is *NP*-hard but open to the sense of the hardness. Particularly, we observe that the complexity status of the special case of this problem with equal-length jobs remains to be open, too. The importance of the complexity of this problem to both theoretical and practical scheduling study is explicit. Nevertheless, to the best of our knowledge, there is no article on any case of this problem. Therefore, it is both interesting and challenging to focus our thesis on this untouched field.

1.2 Due date assignment problems

Scheduling problems involving due date assignment are of particular importance to both researchers and practicing managers and have been studied for decades. An extensive survey of the study in this area was given by Cheng and Gupta (1989). In their paper, Cheng and Gupta classified the literature into static and dynamic job shop situations. Under each job shop environment, the literature was categorized into single-machine and multiple machine cases. They concluded that, while the static single machine problems with constant or common due dates had been well researched, very little or no work had been done on the dynamic multiple machine problems with sophisticated due date assignment methods. They also identified and suggested some worthwhile areas, in both static and dynamic job shop situations, for future research.

Since the publication of Cheng and Gupta' survey, a large volume of literature concerning due date assignment has been published. However, because of the complication and difficulty of the analysis of the dynamic models, the new literature has still been focused on the static job shop situation. While limited results in the static multiple machine case have been reported, the study on the static single machine case has been extended to a broader area. The extension follows two main directions: (1) from the simplest CON due date assignment method to more sophisticated methods, such as SLK and PPW, and (2) from the classical view that due dates are treated as decision variables but are assigned to corresponding jobs before the scheduling, to an alternate view that each due date is not associated with a specific job but assigned to a job according to the sequencing result.

An interesting observation from the classical view is that, although release dates are considered in all types of the due date assignment methods defined by Cheng and Gupta (1989), almost all papers in the literature do not involve the models with release dates. To the best of our knowledge, the only exceptions are Gordon (1993) and Cheng and Gordon (1994). The former paper dealt with the optimal assignment of SLK due dates to n jobs and scheduling them on a single machine to minimize the maximum tardiness; while the latter paper involved the same problem but the due dates are assigned to the jobs by two methods in which release dates are not considered. In this thesis, we attempt to extend Gordon and Cheng's study to some other due date assignment models with release dates, namely, the (weighted) number of tardy jobs models and the total (weighted) tardiness models.

1.3 Outline of the thesis

In the remainder of this thesis, we study several single machine scheduling problems with release dates. After a comprehensive literature review in the next chapter, we introduce some definitions and fundamental properties as well as a basic algorithm in Chapter 3. All the definitions, properties and algorithm in Chapter 3 are applicable to the general preemptive single machine scheduling problem with a regular criterion.

In Chapter 4, we study the preemptive single machine total tardiness problem with equal-length jobs, which is posed as open by Baptiste (2000). With the optimality properties and results in Chapter 3, we present a decomposition technique by which an $O(n^2)$ algorithm is constructed for the problem considered. Then, we assert that the problem is polynomially solvable.

In Chapter 5, we study another special case of the preemptive single machine total tardiness problem, which is the case with agreeable release dates and due dates. We first show the *NP*-hardness of this problem by an obvious reduction from the well-known *NP*-hard problem $1||\sum T_i$. Then we investigate the optimality properties and derive a pseudo-polynomial algorithm to solve this problem, thus ascertaining that it is *NP*-hard in the ordinary sense.

In Chapter 6, we focus on the complexity of the due date assignment problems with release dates. The models with CON (Constant) due dates are discussed in detail, followed by the discussion for the SLK and TWK models. All models with a regular criterion are considered. With a few exceptions, the complexity of most of the problems are explicitly determined.

In Chapter 7, we extend the results in the previous chapters to the problem $1||\sum T_i$, which is a special case of the problem that we consider in Chapter 5. An alternative proof of the pseudo-polynomial solvability of this problem is provided, and some polynomially solvable cases are investigated.

In Chapter 8, we present our conclusions and recommendations.

Chapter 2

Literature review

2.1 Introduction

In this chapter, we aim at providing a survey of the literature on classical single machine scheduling problems involving release and due dates. We concentrate on deterministic models where a given set of jobs has to be processed on a single machine that can perform only one job at a time, and where all data required to define a problem instance are known with certainty. Specifically, we consider the models concerning the scheduling of n jobs with release dates on a single machine to minimize a due-date-based criterion. Because of the enormous volume of the literature and the limited space of this thesis, we focus only on the papers contributed to the complexity results. For a known *NP*-hard problem, we refer to the paper with the *NP*-hardness proof (and the paper with pseudo-polynomial algorithm, if the problem is *NP*-hard in the ordinary sense); for a polynomially solvable problem, we refer to the papers with polynomial algorithms and indicate the time complexity of the algorithms.

To define a problem, we adopt the well-known three-field notation '$\alpha|\beta|\gamma$' of Graham *et al.* (1979) and Chen *et al.* (1998), where α, β and γ indicate the machine environment, the job characteristics, and the optimality criterion, respectively. We will extend this notation scheme to reflect the job restrictions and the due date assignment methods precisely. In particular, we borrow the concept of 'agreeable' from Lawler (1977) and Lee *et al.* (1992) to describe the 'similarly ordered' job characteristics, and the idea of Koulamas and Kyparisis (2001) to indicate such agreeable features by bracketing the agreeable items.

In a single machine scheduling problem, each job i is characterized by the following data:

- a processing time p_i;
- a release date r_i;
- a due date d_i; and
- a weight w_i.

Given a schedule, for each job i, we can compute:

- Completion time C_i;
- Lateness $L_i = C_i - d_i$;
- Tardiness $T_i = \max\{0, C_i - d_i\}$;
- Unit penalty U_i=1 if $C_i > d_i$ and U_i=0 otherwise; and
- A regular (non-decreasing) cost function $f_i = f(C_i)$.

Some regular due-date-based criteria involve the minimization of:

- $L_{max} = \max_i L_i$: the maximum lateness;
- $f_{max} = \max_i f_i$: the maximum cost;
- $\sum(w_i)U_i$: the (weighted) number of tardy jobs;
- $\sum(w_i)T_i$: the total (weighted) tardiness; and
- $\sum f_i = \sum f(C_i, d_i)$: the total due-date-based cost.

In the three-field descriptor '$\alpha|\beta|\gamma$', we will use α=1 to define the single machine environment, $\gamma \in \{L_{max}, f_{max}, \sum(w_i)U_i, \sum(w_i)T_i, \sum f_i\}$ the optimality criterion, and $\beta \subseteq \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5\}$ the job characteristics as below.

- $\beta_1 \in \{o, r_i, (r_i, d_i), (r_i, d_i)^=, (r_i, -d_i), [r_i, d_i]^N,$
  $(r_i, d_i, p_i), (r_i, d_i, p_i, -w_i), (r_i, p_i, -w_i), (\bar{r}_i, d_i)\}$
  - $\beta_1 = o$: no release dates are specified;
  - $\beta_1 = r_i$: jobs have release dates;
  - $\beta_1 = (r_i, d_i)$: release dates and due dates are agreeable, in the sense that $r_i \le r_j$ implies $d_i \le d_j$;
  - $\beta_1 = (r_i, d_i)^=$: release dates and due dates are strictly agreeable, in the sense that $r_i < r_j$ implies $d_i \le d_j$ and $r_i = r_j$ implies $d_i = d_j$;
  - $\beta_1 = (r_i, -d_i)$: release dates and due dates are reversely agreeable, in the sense that $r_i \le r_j$ implies $d_i \ge d_j$;
  - $\beta_1 = [r_i, d_i]^N$: the intervals $[r_i, d_i]$ are nested, in the sense that $[r_i, d_i] \cap [r_j, d_j]$ is either Φ or $[r_i, d_i]$, or $[r_j, d_j]$;
  - $\beta_1 = (r_i, d_i, p_i)$: all release dates, due dates and processing times are agreeable, in the sense that $r_i \le r_j$ implies $d_i \le d_j$ and $p_i \le p_j$;
  - $\beta_1 = (r_i, d_i, p_i, -w_i)$: all release dates, due dates and processing times are agreeable, and weights are reversely agreeable, in the sense that $r_i \le r_j$ implies $d_i \le d_j$, $p_i \le p_j$ and $w_i \ge w_j$;
  - $\beta_1 = (r_i, p_i, -w_i)$: release dates and processing times are agreeable, and weights are reversely agreeable, in the sense that $r_i \le r_j$ implies $p_i \le p_j$ and $w_i \ge w_j$;
  - $\beta_1 = (\bar{r}_i, d_i)$: modified release dates and due dates are agreeable, in the sense that $\bar{r}_i \le \bar{r}_j$ implies $d_i \le d_j$ in case with precedence constraints, where the modified release date $\bar{r}_i = r_i$ if $B(i) = \phi$, and $\bar{r}_i = \max\{r_i, \max_{k \in B(i)} (\bar{r}_k + p_k)\}$ if $B(i) \neq \phi$ (B(i) is the set of all immediate predecessors of job i in N=1,....,n).

- $\beta_2 = \{o, d_i = d, d_i \in \{d'_1, ..., d'_m\}, d_i = r_i + d,$
  $d_i = r_i + p_i + d, d_i = r_i + kp_i, d_i = k_1 p_i + k_2, d_i = k_1 p_i^{k_2}\}$
  - $\beta_2 = o$: due dates are arbitrary;
  - $\beta_2 = d_i = d$: due dates are common for all jobs, where d is a constant;
  - $\beta_2 = d_i \in \{d'_1, ..., d'_m\}$: there are m, $1 < m < n$, distinct due dates;
  - $\beta_2 = d_i = r_i + d$: due dates are determined by the CON method, that is, all jobs are given exactly the same flow allowance;
  - $\beta_2 = d_i = r_i + p_i + d$: due dates are determined by the SLK method, that is, jobs are given flow allowance that reflect equal slacks;
  - $\beta_2 = d_i = r_i + kp_i$: due dates are determined by the TWK method, that is, due dates are based on total work content, where k is a constant;
  - $\beta_2 = d_i = k_1 p_i + k_2$: due dates are determined by the PPW (processing-time-plus-wait) method, where k_1 and k_2 are constant;
  - $\beta_2 = d_i = k_1 p_i^{k_2}$: due dates are determined by the TWK-power (total-work-content-power) method.
- $\beta_3 \in \{o, pmtn\}$
  - $\beta_3 = o$: no preemption is allowed;
  - $\beta_3 = pmtn$: preemption is allowed.
- $\beta_4 = \{o, chain, tree, prec\}$
  - $\beta_4 = o$: no precedence constraints are specified;
  - $\beta_4 = chain$: precedence constraints on jobs are defined where each vertex has outdegree and indegree at most one;
  - $\beta_4 = tree$: precedence constraints on jobs are defined by a rooted intree or outtree;
  - $\beta_4 = prec$: jobs have arbitrary precedence constraints.

- $\beta_5 = \{o, p_i = 1, p_i = p, (p_i, -w_i)\}$
  - $\beta_5 = o$: processing times are arbitrary;
  - $\beta_5 = p_i = 1$: all jobs have unit processing times;
  - $\beta_5 = p_i = p$: all jobs have equal processing times;
  - $\beta_5 = (p_i, -w_i)$: processing times and weights are reversely agreeable.

The rest of this chapter is organized as follows. Problems with a criterion of the maximum lateness, the (weighted) number of tardy jobs and the total (weighted) tardiness are reviewed in Sections 2.2, 2.3 and 2.4, respectively. The due date assignment problems are considered in Section 2.5. Some potential research opportunities are presented in Section 2.6.

2.2 Maximum lateness

Although the general problem $1|r_i|L_{max}$ is shown to be strongly *NP*-hard by Lenstra *et al.* (1977), its preemptive counterpart is polynomially solvable, even in case with arbitrary precedence constraints. Using a generalized earliest due date (EDD) algorithm in $O(n \log n)$ time, Horn (1972) constructed an optimal solution for $1|r_i, pmtn|L_{max}$. Later, Lageweg *et al.* (1976) proposed an $O(n \log n)$ algorithm for the problem $1|r_i, pmtn, tree|L_{max}$. After that, Baker *et al.* (1983) and Gordon and Tanaev (1983) independently generalized Lawler's approach for the non-preemptive problem $1|prec|f_{max}$ to $1|r_i, pmtn, prec|f_{max}$. The algorithm solves $|r_i, pmtn, prec|L_{max}$ in $O(n^2)$ time.

In the case of unit-length jobs, $1|r_i, p_i = 1|L_{max}$ can be solved in $O(n)$ time (Frederickson 1983). Even with arbitrary precedence constraints, the problem $1|r_i, prec, p_i = 1|L_{max}$ can be solved by an $O(n^2)$ algorithm presented by Lageweg *et al.* (1976). In the case of equal-length jobs, the problem $1|r_i, prec, p_i = p|L_{max}$

is still polynomially solvable (Simmon 1978). On the other hand, according to the generalized EDD rule presented by Horn, a preemption occurs at a release date in the case that the newly released job has a smaller due date than that of the job currently being processed, a case that never exist if all release dates and due dates are agreeable. So, the EDD rule, which is equivalent to the earliest release date (ERD) rule in this special case, is optimal for the problem $1|(r_i, d_i)|L_{max}$. Moreover, Gordon (1993) indicated that, if the modified release dates and due dates are agreeable, the algorithm introduced by Baker *et al.* (1983) and Gordon and Tanaev (1983) for the problem $1|r_i, pmtn, prec|f_{max}$ solves $1|(\bar{r}_i, d_i), prec|L_{max}$ and $1|(\bar{r}_i, d_i), tree|L_{max}$ in $O(n^2)$ and $O(n \log n)$ time, respectively.

The detailed complexity results of single machine maximum lateness problems with release dates are summarized in Table 2.1. As a rule, if a problem with no precedence constraints is shown as strongly *NP*-hard, the counterparts with any precedence constraints must be strongly *NP*-hard and are not included in the table; if a problem with arbitrary precedence constraints can be solved by a polynomial algorithm, all counterparts with simpler or no precedence constraints also can be solved by this algorithm and are not included, unless a better algorithm exists. Besides, we use the following notations.

- s-*NP*: strongly *NP*-hard;
- o-*NP*: *NP*-hard in the ordinary sense;
- *NP*: *NP*-hard but open to the sense of hardness;
- open: open to the complexity.

2.3 Number of tardy jobs

As the problem $1|chain, p_i = 1|\sum U_i$ has been proved strongly *NP*-hard by Lenstra and Rinnooy Kan (1980), the counterpart with release dates must be strongly *NP*-hard, too. So, the literature on this class is focused on the problems with no

Table 2.1: Single machine maximum lateness problems with release dates

Problem	Status	Reference
$1\|r_i\|L_{max}$	s-*NP*	Lenstra *et al.* (1977)
$1\|r_i, p_i = 1\|L_{max}$	$O(n)$	Frederickson (1983)
$1\|r_i, prec, p_i = 1\|L_{max}$	$O(n^2)$	Lageweg *et al.* (1976)
$1\|r_i, prec, p_i = p\|L_{max}$	$O(n^2 \log n)$	Simons (1978)
$1\|r_i, pmtn\|L_{max}$	$O(n \log n)$	Horn (1972)
$1\|r_i, pmtn, tree\|L_{max}$	$O(n \log n)$	Lageweg *et al.* (1976)
$1\|r_i, pmtn, prec\|L_{max}$	$O(n^2)$	Baker *et al.* (1983)
		Gordon & Tanaev (1983)
$1\|(r_i, d_i)\|L_{max}$	$O(n \log n)$	ERD rule
$1\|(\bar{r}_i, d_i), tree\|L_{max}$	$O(n \log n)$	Gordon (1993)
$1\|(\bar{r}_i, d_i), prec\|L_{max}$	$O(n^2)$	Gordon (1993)

precedence constraints. In non-preemptive case, $1|r_i|\sum U_i$ is strongly *NP*-hard since $1|r_i|L_{max}$ is already strongly *NP*-hard (Lenstra *et al.* 1977). In preemptive case, the weighted problem $1|r_i, pmtn|\sum w_iU_i$ is *NP*-hard in the ordinary sense and can be pseudo-polynomially solved by an $O(n^3W^2)$ algorithm presented by Lawler (1990), where W is the sum of the integer job weights. Consequently, the unweighted counterpart $1|r_i, pmtn|\sum U_i$ can be polynomially solved in $O(n^5)$ time. Recently, Baptiste (1999a) proposed a new dynamic programming algorithm whose time complexities is $O(n^4)$ for the unweighted problem.

With unit-length jobs, $1|r_i, p_i = 1|\sum w_iU_i$ can be solved in $O(n^3)$ time as an assignment problem. However, Lawgeweg and Lawler (1995) provided a more efficient algorithm to solve this problem in $O(n^2)$ time. In the case of equal-length jobs, Baptiste (1999b) presented an algorithm based on dynamic programming to solve $1|r_i, p_i = p|\sum w_iU_i$ and $1|r_i, pmtn, p_i = p|\sum w_iU_i$ in $O(n^7)$ and $O(n^{10})$ time, respectively. Kise *et al.* (1978) firstly showed that in the case of agreeable release dates and due dates, the unweighted problem $1|(r_i, d_i)|\sum U_i$ can be solved in $O(n^2)$ time. Lawler (1990) mentioned that in such an agreeable case, his dynamic programming algorithm solves $1|(r_i, d_i)|\sum w_iU_i$ and $1|(r_i, d_i), pmtn|\sum w_iU_i$ in $O(nW)$ time, and $1|(r_i, d_i)|\sum U_i$ and $1|(r_i, d_i), pmtn|\sum U_i$ in $O(n^2)$ time, respectively. In another paper, Lawler (1994) generalized the Moore-Hodgson algorithm that solves the problem $1||w_iU_i$ in $O(nW)$ time, to solve both $1|(r_i, d_i)|\sum U_i$ and $1|(r_i, d_i), pmtn|\sum U_i$ in $O(n \log n)$ time. However, we note that even in this special agreeable case, both $1|(r_i, d_i)|\sum w_iU_i$ and $1|(r_i, d_i), pmtn|\sum w_iU_i$ are still *NP*-hard, which is indicated by the *NP*-hardness of the problem $1||\sum w_iU_i$ whose jobs obviously have agreeable release dates and due dates. In case with more complicated restrictions, Tanaev and Gordon (1983) indicated that $1|(r_i, d_i, p_i, -w_i)|\sum w_iU_i$ can be solved in $O(n \log n)$ time; Lawler (1994) showed that $1|(r_i, -d_i), pmtn|\sum w_iU_i$ is pseudo-polynomially solvable in $O(nW)$ time, and both $1|(r_i, p_i, -w_i), pmtn|\sum w_iU_i$ and $1|[r_i, d_i]^N, pmtn, (p_i, -w_i)|\sum w_iU_i$ can be solved in $O(n \log n)$ time. The detailed complexity results of single machine number of tardy jobs problems with release dates are summarized in Table 2.2.

2.4 Total tardiness

Since the strong *NP*-hardness of the problem $1|chain, p_i = 1|\sum T_i$ is shown by Leung and Young (1990), its counterpart with release dates must be strongly *NP*-hard. Thus, the literature on this class also focuses on the problems with no precedence constraints. Again, $1|r_i|\sum T_i$ is strongly *NP*-hard following the strong *NP*-hardness of $1|r_i|L_{max}$ (Lenstra *et al.* 1977). Chu (1992) gave a proof of the *NP*-hardness of $1|r_i, pmtn|\sum T_i$, but did not decide whether it is in the strong sense or in the ordinary sense. However, the weighted problem $1|r_i, pmtn|\sum w_iT_i$ is strongly *NP*-hard (Labetoulle *et al.* 1984).

The problem $1|r_i, p_i = 1|\sum w_iT_i$ can be solved in $O(n^3)$ time, as it also can

Table 2.2: Single machine number of tardy jobs problems with release dates

Problem	Status	Reference
$1\|r_i\|\sum U_i$	s-*NP*	Lenstra *et al.* (1977)
$1\|r_i, pmtn\|\sum U_i$	$O(n^5)$	Lawler (1990)
	$O(n^4)$	Baptiste (1999a)
$1\|r_i, pmtn\|\sum w_iU_i$	$O(n^3W^2)^\bullet$	Lawler (1990)
$1\|r_i, p_i = 1\|\sum w_iU_i$	$O(n^2)$	Lageweg & Lawler (1975)
$1\|r_i, p_i = p\|\sum w_iU_i$	$O(n^7)$	Baptiste (1999b)
$1\|r_i, pmtn, p_i = p\|\sum w_iU_i$	$O(n^{10})$	Baptiste (1999b)
$1\|(r_i, d_i)\|\sum U_i$	$O(n^2)$	Kise *et al.* (1978)
	$O(n \log n)$	Lawler (1990,1994)
$1\|(r_i, d_i), pmtn\|\sum U_i$	$O(n \log n)$	Lawler (1990,1994)
$1\|(r_i, d_i)\|\sum w_iU_i$	$O(nW)^\bullet$	Lawler (1990)
$1\|(r_i, d_i), pmtn\|\sum w_iU_i$	$O(nW)^\bullet$	Lawler (1990)
$1\|(r_i, -d_i), pmtn\|\sum w_iU_i$	$O(nW)^\bullet$	Lawler (1994)
$1\|(r_i, d_i, p_i, -w_i)\|\sum w_iU_i$	$O(n \log n)$	Tanaev & Gordon (1983)
$1\|(r_i, p_i, -w_i), pmtn\|\sum w_iU_i$	$O(n \log n)$	Lawler (1994)
$1\|[r_i, d_i]^N, pmtn, (p_i, -w_i)\|\sum w_iU_i$	$O(n \log n)$	Lawler (1994)

$^\bullet$: o-*NP*

be transformed to an assignment problem. In case of equal-length jobs, Baptiste (2000) presented an $O(n^7)$ algorithm to solve some single machine scheduling problems including $1|r_i, p_i = p|\sum T_i$. Nevertheless, according to Brucker and Knust (2002), the complexity status of all the problems $1|r_i, pmtn, p_i = p|\sum T_i$, $1|r_i, p_i = p|\sum w_iT_i$ and $1|r_i, pmtn, p_i = p|\sum w_iT_i$ are open to date. In case with agreeable release dates and due dates, $1|(r_i, d_i)|\sum T_i$ is proved strongly *NP*-hard by Koulamas and Kyparisis (2001) recently, but both $1|(r_i, d_i), pmtn|\sum T_i$ and $1|(r_i, d_i), pmtn|\sum w_iT_i$ are still open to date. The detailed complexity results of single machine total tardiness problems with release dates are summarized in Table 2.3.

Table 2.3: Single machine total tardiness problems with release dates

Problem	Status	Reference
$1\|r_i\|\sum T_i$	s-*NP*	Lenstra *et al.* (1977)
$1\|r_i, pmtn\|\sum T_i$	*NP*	Chu (1992)
$1\|r_i, pmtn\|\sum w_iT_i$	s-*NP*	Labetoulle *et al.* (1984)
$1\|r_i, p_i = 1\|\sum w_iT_i$	$O(n^3)$	Assignment-problem
$1\|r_i, p_i = p\|\sum T_i$	$O(n^7)$	Baptiste (2000)
$1\|r_i, p_i = p\|\sum w_iT_i$	**open**	Brucker & Knust (2002)
$1\|r_i, pmtn, p_i = p\|\sum T_i$	**open**	Brucker & Knust (2002)
$1\|r_i, pmtn, p_i = p\|\sum w_iT_i$	**open**	Brucker & Knust (2002)
$1\|(r_i, d_i)\|\sum T_i$	s-*NP*	Koulamas & Kyparisis (2001)
$1\|(r_i, d_i), pmtn\|\sum T_i$	**open**	
$1\|(r_i, d_i), pmtn\|\sum w_iT_i$	**open**	

2.5 Due date assignment

As the first survey focused on the due date assignment problems, Cheng and Gupta (1989) covered the research results in the literature up to then. According to Cheng and Gupta, no paper published before the 1990s dealt with the due date assignment problems with release dates. Following Cheng and Gupta' survey, a large volume of literature concerning due date assignment has been published. In two very recent papers, Gordon *et al.* (2002a, 2002b) provided comprehensive surveys of due date assignment research on CON due date assignment model, and SLK, TWK as well as other due date assignment models, respectively. Their surveys showed that, compared with the results summarized in Cheng and Gupta (1989), a large amount of effort has been devoted to the more complicated models such as SLK and TWK. Moreover, an alternative view that the due dates are specified according to the position in which a job is completed, rather than the identity of that specific job, is proposed and has been a hot research topic.

Nevertheless, as indicated in the last section, Gordon (1993) and Cheng and Gordon (1994) are the only papers in the literature that take release dates into considerations, with very few exceptions dealing with the positional due date models or the computer simulation models, both of which are beyond the scope of our research. Gordon (1993) considered the preemptive SLK due date assignment problem with precedence constraints. The objective is to find an optimal schedule S^* and an optimal slack d^* allowance that jointly minimize a penalty function given by

$$f(S,d) = \alpha d + \max_{i \in N} T_i$$

where $\alpha \geq 0$ is the cost per unit time of slack. It is shown that the schedule obtained by applying the algorithm proposed by Baker *et al.* (1983) and Gordon and Tanaev (1983) for the problem $1|r_i, pmtn, prec|f_{max}$, which is called Algorithm GT-BLLR in a later paper (Cheng and Gordon 1994), is optimal independently of the value of d, and that the problem is solvable in $O(n^2)$ time in the case of arbitrary precedence constraints and $O(n \log n)$ time in the case of tree-like precedence constraints.

Cheng and Gordon (1994) studied the same problem as that in Gordon (1993), but with PPW and TWK-power due date assignment methods. The objective is to find an optimal schedule S^* and optimal values k_1^* and k_2^* that jointly minimize a penalty function given by

$$f(S, k_1, k_2) = \Phi(k_1, k_2) + \alpha \max_{i \in N} T_i$$

where $\Phi(k_1, k_2) \geq 0$, a nondecreasing convex functions of k_1 and k_2 with $\Phi(0,0) =$

0, is the cost of assigning due dates and $\alpha \geq 0$ is the cost per unit tardiness incurred by the most tardy job. It is shown that the schedule obtained by applying Algorithm GT-BLLR is independently optimal of the values k_1 and k_2, and thus, the scheduling problem is solvable in $O(n^2)$ time in the case of arbitrary precedence constraints and $O(n \log n)$ time in the case of tree-like precedence constraints. It is also showed that, with PPW due date assignment method, the due date assignment problem is solvable in $O(n^2)$ time if $\Phi(k_1, k_2) = \Phi_1(k_1) + \beta_2 k_2$, where $\Phi_1(k_1)$ is a nondecreasing convex function with $\Phi_1(0) = 0$ and $\beta_2 \geq 0$. The detailed complexity results of single machine due date assignment problems with release dates are summarized in Table 2.4.

Table 2.4: Single machine due date assignment problems with release dates

Problem	Status	Reference
$1\|r_i, d_i = r_i + p_i + d, pmtn, tree\|\alpha d + \max_{i \in N} T_i$	$O(n \log n)$	I
$1\|r_i, d_i = r_i + p_i + d, pmtn, prec\|\alpha d + \max_{i \in N} T_i$	$O(n^2)$	I
$1\|r_i, d_i = k_1 p_i^{k_2}, pmtn, tree\|\Phi(k_1, k_2) + \alpha \max_{i \in N} T_i$	$O(n \log n)$	II
$1\|r_i, d_i = k_1 p_i^{k_2}, pmtn, prec\|\Phi(k_1, k_2) + \alpha \max_{i \in N} T_i$	$O(n^2)$	II
$1\|r_i, d_i = k_1 p_i + k_2, pmtn, tree\|\Phi(k_1, k_2) + \alpha \max_{i \in N} T_i$	$O(n \log n)$	II
$1\|r_i, d_i = k_1 p_i + k_2, pmtn, prec\|\Phi(k_1, k_2) + \alpha \max_{i \in N} T_i$	$O(n^2)$	II
$1\|r_i, d_i = k_1 p_i + k_2, pmtn, tree\|\Phi(k_1) + \beta_2 k_2 + \alpha \max_{i \in N} T_i$	$O(n^2)$	II

Reference: I, Gordon (1993); II, Cheng and Gordon (1994)

2.6 Potential research opportunities

The review confirms our argument that single machine scheduling problems could be rather difficult, especially in case with release dates and due-date-based criterion. Despite considerable research effort expended on studying these models, there still exist rich opportunities in this area for further research. The following are some worthy future research topics.

1. Total cost models with equal-length jobs

Although some of the single machine scheduling problems with arbitrary release dates and equal-length jobs are polynomially solvable, the review shows that a few of them are still open to date, as also indicated by Brucker and Knust (2002). It seems very difficult to determine the complexity of these open problems. On the one hand, the arbitrary release dates bring about preemptions and/or idleness in an optimal schedule, both of which increase the difficulty of finding an optimal solution. On the other hand, the equal processing times prevent us from constructing a reduction from a known *NP*-hard problem, such as the 2-partition and 3-partition problems, in most cases. Undoubtedly, all these open problems are worthy of research. Besides, the time complexity of Baptiste' (1999b, 2000) two algorithms for some equal-length jobs problems is not so satisfactory. Better algorithms or improvement on Baptiste's algorithms are worthy to be pursued.

2. Total tardiness models with agreeable release dates and due dates

Since the arbitrary due dates models are intractable in most cases, it is both reasonable and valuable to investigate the models with restricted due dates. Lawler, together with some other researchers, have made an intensively study on the (weighted) number of tardy jobs models with several different agreeable features. In spite of the fact that all the agreeable cases in Tables 2.1 and 2.2 are polynomially or pseudo-polynomially solvable, the corresponding total tardiness problems are either proved strongly *NP*-hard or remain to be open. It should be interesting to study the open problems and the more restricted cases of the known *NP*-hard problems.

3. Due date assignment models

Scheduling problems involving due date assignment are of permanent interest. However, hardly any prior research has considered the release dates. Besides the single machine models with a regular criterion that we have reviewed, there are plenty of other more complicated models worthy of study. Gordon *et al.* (2002a, 2002b) showed that, because of the importance of the just-in-time (JIT) systems in inventory management, most of the papers on CON due date assignment problems involved earliness, a non-regular criterion, but did not consider release dates. On the other hand, though a lot of the papers in the literature paid attention to the multiple machine models, none of them considered release dates. In conclusion, due date assignment problems with release dates are rarely touched. It is timely to explore this area with rich opportunities.

Chapter 3

Preemptive scheduling with a regular criterion

3.1 Introduction

As shown in Brucker and Knust's (2002) classification as well as many scheduling textbooks, single machine scheduling problems without release dates are relatively easy to tackle. Except the problem $1||\sum w_iT_i$, all the others are either polynomially or pseudo-polynomially solvable. Moreover, even the strongly $\mathcal{NP}$-hard problem $1||\sum w_iT_i$ can be dealt with through dynamic programming (DP), a very useful technique that can be applied to both polynomially solvable problems and $\mathcal{NP}$-hard problems. But, if preemption is not allowed, this widely used technique is no longer applicable to the single machine problems with release dates, even for minimizing a regular criterion. Consequently, with release dates, all the non-preemptive single machine scheduling problems with a regular criterion except the makespan are strongly $\mathcal{NP}$-hard. Nevertheless, dynamic programming is valid for the preemptive counterparts of these problems, since in the preemptive case, the makespan of a problem with a regular criterion is schedule independent, which implies that the start and ending times are deterministic in each recursion and the DP approach is feasible. All based on dynamic programming technique, Algorithm GT-BLLR for the problem $1|r_i, pmtn, prec|f_{max}$ is presented by Baker *et al.* (1983) and Gordon and Tanaev (1983) independently; an $O(n^3W^2)$ algorithm for $1|r_i, pmtn|\sum w_iU_i$ is proposed by Lawler (1990); an $O(n^4)$ algorithm for $1|r_i, pmtn|\sum U_i$, which requires $O(n^5)$ time by Lawler' algorithm, is provided by Baptiste (1999a).

It is also shown in Brucker and Knust's (2002) classification that all the open problems in the single machine class except $1|r_i, p_i = p|\sum w_iT_i$ involve preemption. To conquer these preemptive open problems, dynamic programming is a helpful technique. In this chapter, we intend to present some useful definitions for the analysis of the preemptive single machine scheduling problems with a regular criterion, and identify optimality properties for these problems, especially for the total tardiness problems. Besides, we propose a dynamic programming algorithm. Unless explicitly stated, the definitions, properties and algorithm hold for all preemptive single machine scheduling problems with a regular criterion, such as $1|r_i, pmtn|f_{max}$ and $1|r_i, pmtn|\sum f_i$.

The rest of this chapter is organize as follows. Several definitions are provided in the next section. In Section 3.3, we present an algorithm based on dynamic programming. In Section 3.4, we identify some optimality properties, with our conclusions given in Section 3.5.

3.2 Definitions

To facilitate presentation, we recall two definitions in the literature first, and then introduce some new definitions.

Definition 3.1 (Pinedo 1995) *A feasible schedule is called* **nondelay** *if no machine is kept idle when there is an operation available for processing.*

As indicated in Pinedo (1995), nondelay schedules are dominant for all preemptive models with a regular criterion.

Definition 3.2 (Baker *et al.* 1983) *A* **block** $B \subseteq N$ *is defined as a minimal set of jobs processed without idleness from* $r(B) = \min_{i \in B}\{r_i\}$ *until* $t(B) = r(B)+p(B)$ *(*$p(B) = \sum_{i \in B} p_i$*), such that each job* $i \notin B$ *is either completed not later than* $r(B)$ *(*$C_i \le r(B)$*) or not released before* $t(B)$ *(*$r_i \ge t(B)$*).*

Definition 3.3 *An* **ERD schedule** *is defined as a nondelay schedule in which all jobs are scheduled by the* **earliest release date (ERD)** *rule.*

By Definition 3.3, even if the jobs are preemptive, no preemption is created in an ERD schedule. Nevertheless, an ERD schedule is helpful for finding an optimal schedule for a preemptive problem.

According to Baker *et al.* (1983), the makespan of a preemptive single machine scheduling problem with a regular criterion can be determined in advance by scheduling the jobs by the ERD rule. This ERD schedule naturally decomposes into some blocks, each of which can be considered separately. So, the scheduling of an overall problem is decomposed into some sub-problems, each of which involves the scheduling of a block. From here onward, we focus our analysis on the scheduling of a block B in the initial ERD schedule.

Consider any nondelay schedule for block B. By Definitions 3.1 and 3.2, all jobs in B are scheduled within $[r(B), t(B)]$ without idleness. Since preemption is allowed, a job $i \in B$ may be divided into a number of pieces separated by other jobs. Suppose we postpone all pieces except the last of a preempted job as late as possible. After the postponing, the completion time of this job is unchanged. At the same time, the completion of any other job is either advanced or unaffected. Applying this procedure to all preempted jobs, we get a new schedule not worse than the old one. In the new schedule, no job's completion can be advanced without postponing the completion of any other job. Such a schedule is called p-active.

Definition 3.4 *A preemptive schedule is called* **p-active** *if no job's completion can be advanced without postponing the completion of any other job.*

It is easily seen that a p-active schedule has to be nondelay. However, the reverse is not necessarily true. For example, consider the scheduling of two jobs J_1 and J_2 with $r_1 = 0$, $r_2 = 1$ and $p_1 = p_2 = 2$. The schedule in which J_1 is scheduled within the intervals $[0, 1]$ and $[2, 3]$, while J_2 within the intervals $[1, 2]$ and $[3, 4]$, is nondelay, but not p-active, since we can re-schedule J_1 and J_2 so that J_1 is scheduled within the interval $[0, 2]$ and J_2 within the interval $[2, 4]$, with the completion of J_1 advanced but no delay to J_2. So, p-active schedules are dominant for all nondelay schedules, and consequently, also dominant for all preemptive models with a regular criterion. From here onward, only p-active schedules are considered.

Definition 3.5 *An* **optimal ending job** *of a set $M \in N$ is defined as a job $k \in M$ that can be completed later than any job $j \in M \setminus \{k\}$ in an optimal schedule.*

By Definitions 3.4 and 3.5, if a job is completed at $t(B)$ in a p-active schedule, it must have been postponed as late as possible and is scheduled within the interval(s) where any other job is either completed or unreleased. It is easy to see that a job set may have more than one optimal ending jobs.

Definition 3.6 *A* **subblock** *$B_i \subset B$ is defined as a minimal subset of a block B that can be defined as a block when some jobs $j \in B$ are removed from B; a subblock B_i is call* **optimal** *if there exists an optimal schedule in which all jobs in B_i are scheduled within $[r(B_i), t(B_i)]$.*

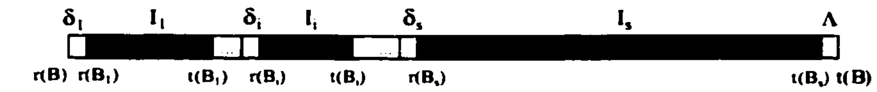

Figure 3.1: Block B after the decomposition by a subset U

Consider the initial ERD schedule for block B. Suppose we remove all jobs in a selected subset $U \subset B$ from the schedule and advance the rest jobs as early as possible by the ERD rule. The new ERD schedule for the set $B \setminus U$ decomposes into some subblocks B_i, i=1,...,s. Such a procedure is called a *decomposition* for block B (by U). The result of a decomposition is shown in Figure 3.1.

Definition 3.7 *A* **decomposition** *for a block B is defined as a procedure during which all jobs in a subset $U \subset B$ is removed from the initial ERD schedule for B and the rest jobs in the subset $B \setminus U$ are re-scheduled by the ERD rule.*

After a decomposition by U, block B is artificially decomposed into a number of subsets: U and B_is. At the same time, the interval $[r(B), t(B)]$ is partitioned into some smaller intervals: I_i occupied by B_i, i=1,...,s, and the rest that come to be idle because of the remove of U. The notations are given as follows.

Notations:

For all i=1,...,s,

- $r(B_i) = \min_{j \in B_i}\{r_j\}$, the start of I_i;
- $t(B_i) = r(B_i) + p(B_i)$, the end of I_i ($p(B_i) = \sum_{j \in B_i} p_j$);
- $I_i = [r(B_i), t(B_i)]$, the i-th non-idle intervals ($I_0 = [r(B), r(B)]$);
- $\delta_i = [t(B_{i-1}), r(B_i)]$, the idle interval between I_{i-1} and I_i;
- $\Lambda = [t(B_s), t(B)]$, the last idle interval following I_s;
- $\Delta = \cup_{i=1}^{s} \delta_i$, the union of all idle intervals except Λ.

Whenever there is no ambiguity, we also use δ_i, Δ and Λ to denote the length of the interval of δ_is, Δ and Λ, respectively. Then, we have $\Delta + \Lambda = p(U) = \sum_{j \in U} p_j$.

We note that δ_i=0 occurs when $t(B_{i-1}) = r(B_i)$, but Λ=0 never occur, according to Definition 3.2. So, we always have $\Lambda > 0$ and $0 \leq \Delta < \sum_{j \in U} p_j$.

Consider the result of a decomposition for block B by U. If there exists an optimal schedule in which all jobs in B_i are scheduled within I_i, i=1,...,s, we always can get an optimal schedule by optimally scheduling all jobs in U within the idle intervals, and re-scheduling B_i within I_i, i=1,...,s, separately in an optimal way. In such a case, we say the decomposition result is *optimal*. Hence, if a decomposition result can be proved optimal, the sub-problem of the scheduling of block B is decomposed into s+1 smaller sub-problems: the scheduling of U within $\Delta \cup \Lambda$ and B_i within I_i, i=1,...,s.

3.3 A dynamic programming algorithm

Consider a special decomposition for block B by a subset with only one job $k \in B$ that is known as an optimal ending job of B in advance. By Definitions 3.4 and 3.5, there exists an optimal schedule in which job k is scheduled within $\Delta \cup \Lambda$, the interval(s) where any job in $B \setminus \{k\}$ is either completed or unreleased, and completed at $t(B)$. At the same time, all jobs in B_i are scheduled within I_i, i=1,...,s. So, the decomposition result is optimal and the sub-problem of the scheduling of block B decomposes into s+1 smaller sub-problems: job k within $\Delta \cup \Lambda$ and B_i within I_i, i=1,...,s. As $\Delta \cup \Lambda = p_k$, job k can be scheduled within the idle intervals immediately after the decomposition. Thus, an optimal schedule for block B can be gained by repeatedly applying the decomposition procedure to each of the subblocks. The following algorithm, a generalization of Algorithm GT-BLLR, can be applied to all preemptive single machine scheduling problems with a regular criterion.

Algorithm SMPP (Single Machine Preemptive Problem)

```
1. Index the jobs by the ERD rule.
2. Schedule the jobs by the ERD rule.
3. For each block B,
    3.1. Identify an optimal ending job $k \in B$.
    3.2. Decompose B by job $k$.
    3.3. Schedule job $k$ within $\Delta \cup \Lambda$.
    3.4. For each of the subblocks $B_i$, repeat Step 3, unless $|B_i| = 1$.
```

In Algorithm SMPP, Steps 1 and 2 require $O(n \log n)$ and $O(n)$ time, respectively. For each block B, Step 3.2 can be done in $O(|B|)$ time, since the jobs are already in ERD order; Step 3.3 requires no additional time; so the time requirement is determined by Step 3.1.

In Algorithm GT-BLLR, the condition

$$f_k(t(B)) = min_{j \in B}\{f_j(t(B))\} \tag{3.1}$$

is used to identify an optimal ending job of block B in $O(|B|)$ time. So, Algorithm GT-BLLR solves the problem $1|r_i, pmtn|f_{max}$ in $O(n^2)$ time. However, it seems impossible to identify an optimal ending job of a block in polynomial time for many problems in the form of $1|pmtn, r_i|\sum f_i$, such as $1|r_i, pmtn|\sum w_iC_i$, $1|r_i, pmtn|\sum w_iT_i$ and $1|r_i, pmtn|\sum T_i$, all of which have been proved *NP*-hard (Labetoulle *et al.* 1984 and Chu 1992). Nevertheless, the dominance rules introduced in what follows can help to restrict the search for the optimal ending jobs. So, Algorithm SMPP can be an efficient enumerate approach for the *NP*-hard cases. Moreover, it will be shown that the problem $1|r_i, pmtn, p_i = p|\sum T_i$ can be polynomially solved by an $O(n^2)$ algorithm which is similar to but more complicated than Algorithm SMPP.

3.4 Optimality properties

Let s_i denote the start time of a job i in any feasible schedule for a preemptive single machine scheduling problem with a regular criterion. We have $r_i \leq s_i$ for all jobs $i \in N$. We describe the relation between a job pair i, j in a preemptive schedule by the relation between the intervals $I_l = [s_l, C_l]$, l=i,j. For any job pair i, j with $s_i < s_j$, there are three cases of the relation between I_i and I_j:

- $I_i \cap I_j = \Phi$ or s_j, i.e. $s_i < C_i \leq s_j < C_j$, we say job i precedes job j and write as $i \rightarrow j$.
- $I_i \cap I_j = [s_j, C_i]$, i.e. $s_i < s_j < C_i < C_j$, we say jobs i, j overlap and write as $i \rightarrow j \rightarrow i \rightarrow j$.
- $I_i \cap I_j = I_j$. i.e. $s_i < s_j < C_j < C_i$, we say job i embeds job j and write as $i \rightarrow j \rightarrow i$.

In the case of $i \rightarrow j \rightarrow i \rightarrow j$, the schedule cannot be p-active. So, we have the following lemma.

Lemma 3.1 *Any two jobs do not overlap in a p-active schedule.*

In the case of $i \rightarrow j \rightarrow i$, the schedule also cannot be p-active if: (1) $r_j \leq s_i$, or (2) $s_i < r_j$ but a part of job i is scheduled within $[r_j, C_j]$. In other words, in any p-active schedule, $i \rightarrow j \rightarrow i$ implies: (1) $r_i \leq s_i < r_j$, and (2) no portion of job i is scheduled within $[r_j, C_j]$. Thus, we have the following lemmas.

Lemma 3.2 *For any job pair i, j with $r_i = r_j$, either $i \rightarrow j$ or $j \rightarrow i$ holds in a p-active schedule.*

Lemma 3.3 *For any job pair i, j with $r_i < r_j$, in a p-active schedule,*

(1) if $C_i \leq r_j$, $i \rightarrow j$ holds.

(2) if $s_i < r_j < C_i$, either $i \rightarrow j$ or $i \rightarrow j \rightarrow i$ holds.

(3) if $r_j \leq s_i$, either $i \rightarrow j$ or $j \rightarrow i$ holds.

In the rest of this section, we discuss the determination of the precedence relations between two preemptive jobs. Apparently, if two jobs $i, j \in N$ are in different blocks, the precedence relation is obvious and needs no consideration. So, our discussion is focused on a job pair $i, j \in B$ with $r_i \leq r_j$. By Lemmas 3.2 and 3.3, unless job i is completed not later than the release of job j, the precedence relation should be determined. In the case of $r_i = r_j$ or $r_i < r_j \leq s_i$, we should determine which one can be preceded by the other. In the case of $s_i < r_j < C_i$, we use i^- and i^+ to denote the completed and uncompleted part of job i at r_j, respectively, and regard i^+ as an individual job released at r_j. We should determine which of i^+, j can be preceded by the other. We first introduce two notations that will be used.

- $\mathbf{E_i}$: *the earliest start time* of job i.
- $\mathbf{L_i}$: *the latest completion time* of job i.

Both of the concepts E_i and L_i are first presented by Szwarc (1999) but have new meanings here. Consider a case where the set of the jobs that should be scheduled within an interval $[r(B), t]$, $r(B) \leq t \leq t(B)$, has been determined but the scheduling of these jobs is undetermined yet, and a job $i \in B$ is not included in the set. Although no optimal schedule has been gained yet, by assumption, there must exist an optimal schedule in which $t \leq s_i$ holds. So, all schedules with $s_i < t$ can be eliminated without loss of optimality. We call such a time t the earliest start time of job i and write as E_i. Moreover, if another job $j \in B$ can be proved preceding job i later, E_i can be postponed to $E_i = t + p_j$, which means that the schedules with $t \leq s_i < t + p_j$ also can be eliminated. Obviously, E_i is helpful to restrict the search for an optimal schedule. Similarly, L_i is such a time that all schedules with $L_i < C_i$ can be eliminated without loss of optimality and is helpful to restrict the search, too. It is easily seen that the initial value of E_i and L_i for any job $i \in B$ is $E_i = r_i$ and $L_i = t(B)$, respectively.

Now, we present two theorems for scheduling preemptive jobs to minimize $\sum T_i$, which are similar to Emmons' famous theorems for scheduling non-preemptive jobs. As indicated by Emmons (1969), these properties provide only *existential* properties "There exists an optimal schedule with property A" rather than *universal* properties "All optimal schedules have property A". However, all these properties can be accumulated, which means that if "There exists an optimal schedule with property A" and "There exists an optimal schedule with property B", "There exists an optimal schedule with both properties A and B". More description for the accumulative existential properties can be found in Emmons (1969).

For any job pair $i, j \in B$ with $r_i \leq r_j$, if $r_j \leq E_i$ is known, we have the following theorems to determine the precedence relation between job i and job j.

Theorem 3.1 *For any job pair i, j with $r_i \leq r_j \leq E_i$, if:*

(1) $p_i \leq p_j$ and $d_i \leq \max\{E_j + p_j, d_j\}$, or

(2) $p_i \geq p_j$ and $d_j \geq \max\{L_i - p_j, d_i\}$,

there exists an optimal schedule in which $i \rightarrow j$ holds.

Proof. By the definitions of E_i and L_i, there exists an optimal schedule satisfying $r_l \leq E_l \leq s_l < C_l \leq L_l$, $l = i, j$. Consider any optimal schedule S^* which satisfies this inequality. Suppose $j \rightarrow i$ holds in S^*. We have $s_j^* < C_j^* \leq s_i^* < C_i^*$. As $r_i \leq r_j$, we can re-schedule the pair i, j so that job i is started at s_j^* and job j is not started before the completion of job i, without affecting any other jobs. In the new schedule S', $i \rightarrow j$ holds. Let s_l' and C_l' to denote the start and completion time of job l in S', $l = i, j$, respectively. We have $s_i' = s_j^*$, $C_j' = C_i^*$ and $C_i' \leq C_j' - p_j = C_i^* - p_j$. After the re-scheduling, T_j increases by $\delta T_j = \max\{C_i^* - \max\{C_j^*, d_j\}, 0\}$, but T_i decreases by $\delta T_i = \max\{C_i^* - \max\{C_i', d_i\}, 0\}$.

In case (1), as $p_i \leq p_j$, we have $C_i' \leq C_j^* \leq \max\{C_j^*, d_j\}$; as $E_j + p_j \leq s_j^* + p_j \leq C_j^*$ and $d_i \leq \max\{E_j + p_j, d_j\}$ by assumption, we have $d_i \leq \max\{C_j^*, d_j\}$; so, we have $\max\{C_i', d_i\} \leq \max\{C_j^*, d_j\}$.

In case (2), as $C'_i \leq C^*_i - p_j$ and $C^*_i \leq L_i$, we have $C'_i \leq C^*_i - p_j \leq L_i - p_j$; besides, $\max\{L_i - p_j, d_i\} \leq d_j$ by assumption; so, we have $\max\{C'_i, d_i\} \leq d_j \leq \max\{C^*_j, d_j\}$.

In both cases, we have $\delta T_j - \delta T_i \leq 0$ and $\sum_{l \in N} \Delta T_l \leq 0$, where ΔT_l is the change in tardiness of job l, as no other job is affected. On the other hand, since S^* is optimal, $\sum_{l \in N} \Delta T_l \geq 0$. So, only $\sum_{l \in N} \Delta T_l = 0$ holds, which means that S' is optimal too. Hence, we get an optimal schedule in which $i \rightarrow j$ holds. Recall that by Lemmas 3.2 and 3.3, if $r_i \leq r_j \leq s_i$, either $i \rightarrow j$ or $j \rightarrow i$ holds in an optimal schedule. We have an optimal schedule in which $i \rightarrow j$ holds. □

Theorem 3.2 *For any job pair i, j with $r_i \leq r_j \leq E_i$, if*

(1) $p_i \leq p_j$ and $d_i \geq \max\{L_j - p_i, d_j\}$, or

(2) $p_i \geq p_j$ and $d_j \leq \max\{E_i + p_i, d_i\}$,

there exists an optimal schedule in which $j \rightarrow i$ holds.

Proof. The proof can be done by interchanging i and j in the proof of Theorem 3.1. □

On the other hand, suppose job i is started before but uncompleted until r_j. Let E_{i^+} be the earliest start time of i^+. We have either $i^+ \rightarrow j$ or $j \rightarrow i^+$ by Theorems 3.1 and 3.2, in case the corresponding conditions are satisfied. We should note that $r_{i^+} = r_j$, $d_{i^+} = d_i$, $L_{i^+} = L_i$ and $C_{i^+} = C_i$ hold.

Finally, Corollary 3.1 following Theorems 3.1 and 3.2 can be established.

Corollary 3.1 *For any job pair i, j with $p_i = p_j = p$ and $r_i \leq r_j$, there exists an optimal schedule in which*

(1) $i \rightarrow j$ holds, if $d_i \leq d_j$.

(2) $j \rightarrow i$ holds, if $d_i \geq d_j$ and $E_i \geq r_j$.

3.5 Conclusions

In this chapter, we have presented some definitions for the analysis of the preemptive single machine scheduling problems with a regular criterion, and identified some optimality properties for these problems, especially for the total tardiness problems. Besides, we have proposed a dynamic programming algorithm that is generally applicable. The definitions and optimality properties are not only helpful to the analysis of a general preemptive scheduling models, but also vital to the following research in this thesis.

Chapter 4

Preemptive total tardiness problem with equal-length jobs

4.1 Introduction

In the last few decades, many researchers have been interested in the computational complexity status and algorithms for scheduling problems of minimizing total tardiness, subject to various constraints. A summary of the results can be found in the review by Chen *et al.* (1998). Most of these problems have been shown to be either *NP*-hard or polynomially solvable. Despite considerable research effort expended on studying this class of problems, Brucker and Knust (2002) point out that there are still some unsolved problems whose complexity status is open. The purpose of this chapter is to settle the complexity status of one of these open problems by constructing an $O(n^2)$ algorithm for it.

The problem addressed in this chapter can be formally stated as follows: A set of n jobs $N = \{1, ..., n\}$ has to be processed on a single machine that can perform only one job at a time. Each job i has a release date r_i, a processing time p_i, and a due date d_i (without loss of generality, we assume $r_i \leq d_i$). All r_i, p_i and d_i are integers. The processing times are equal, while the release dates and due dates are arbitrary. Preemption is allowed. The objective is to schedule the jobs so as to minimize total tardiness.

To the best of our knowledge, there is no article except Chu (1992) on the problems of preemptive scheduling on a single machine to minimize the total tardiness. The *NP*-hardness of the general problem $1|r_i, pmtn|\sum T_i$ can be reduced by the well known result that $1||\sum T_i$ is *NP*-hard. A detailed reduction can be found in Chu (1992). However, this problem is still open to the sense of *NP*-hardness. That means, neither a pseudo-polynomial algorithm nor a reduction from any known strongly *NP*-hard problem is gained till now. In the mean time, the weighted counterpart of this problem is strongly *NP*-hard since $1|r_i, pmtn|\sum w_iC_i$ is strongly *NP*-hard.

In the case that all jobs are released simultaneously, preemption is unnecessary. The problem is equivalent to $1|p_i = p|\sum T_i$, which can be solved by the EDD rule. In the case that all jobs have a constant due date, the problem can be denoted as $1|r_i, d_i = d, pmtn, p_i = p|\sum T_i$, where d is a constant. Since $d_i = d$ holds for all jobs $i \in N$, it is never necessary to preempt a processing job, of which the rest processing time is certainly less than p, in favor of a newly released job whose processing time is equal to p. Hence, this preemptive problem can be solved by the ERD rule, with no preemption created. In the case that all jobs are released simultaneously and have a constant due date, the problem is equivalent to $1|d_i = d, p_i = p|\sum T_i$ and the jobs can be scheduled in any order.

If preemption is not allowed, the problem should be denoted as $1|r_i, p_i = p|\sum T_i$ and can be solved by an $O(n^7)$ algorithm proposed by Baptiste (2000). However, Baptiste states that the algorithm cannot be applied to the preemptive case, and poses the complexity status of the problem $1|r_i, pmtn, p_i = p|\sum T_i$ as open. Therefore, it is both interesting and challenging to study this problem. In this paper, we will derive an $O(n^2)$ algorithm to solve this problem, thus ascertaining that it is polynomially solvable.

We adopt the concept of *block*, which has been used to solve another similar problem, e.g. $1|r_i, pmtn, prec|f_{max}$ in Baker *et al.* (1983). Our approach is outlined as follows. First, we decompose a preemptive scheduling problem into some sub-problems, each of which involving the scheduling of a block. Then, we provide some properties for an optimal schedule of a block. Finally, we investigate the scheduling of a block with equal-length jobs and derive some properties for designing a solution algorithm for the problem.

The rest of this chapter is organized as follows: In Section 4.2, we present some optimal properties. In Sections 4.3 and 4.4, we introduce a decomposition technique and provide an algorithm in $O(n^2)$ time, respectively. The number of preemptions is discussed in Section 4.5, with some concluding remarks given in Section 4.6.

4.2 Decomposition technique

In the problem addressed, any block consists of equal-length jobs. It is easy to verify that any simple rules and conditions, including Equation 3.1, is not optimal for identifying an optimal ending job for such blocks. Nevertheless, the scheduling of such a block can still be decomposed into some smaller sub-problems, by a special decomposition procedure. The procedure consists of a multiple stages decomposition for a block B into $B_1,...,B_s$ by a subset $U \subset B$ whose job number increases with the stages. After the decomposition in each stage, we examine the precedence relations between the jobs in B_i and U to judge whether B_i is optimal, from i=1 to s. If the answer is positive for all B_is, the result is optimal; once a negative result is got or no result can be got, add a new job to U and decompose block B by the new subset U in the next stage. It will be shown that an optimal result can be gained in at most $|B|$-1 stages. With an optimal result, we should schedule the set U within the idle intervals before repeating the decomposition procedure for the subblocks in the next iteration. So, each iteration for a block/subblock B is composed of two procedures: (1) the multiple stage decomposition, and (2) the scheduling of the set U.

We first present some notations and basic properties. Following the notations for the precedence relations between two jobs, we define the precedence relations between a subblock B_i and a job $k \notin B_i$ as follows.

- If $k \to j$ holds for all jobs $j \in B_i$, we say job k precedes B_i and write as $k \to B_i$;
- If $j \to k$ holds for all jobs $j \in B_i$, we say B_i precedes job k and write as $B_i \to k$;
- If $j \to k^+$ holds for all jobs $j \in B_i$, we say B_i precedes k^+ and write as $B_i \to k^+$, which implies $k \to B_i \to k$.

Lemma 4.1 *For any subblock B_i and a job $k \notin B_i$, if*

(1) $E_k \geq r(B_i)$, and

(2) $d_k \geq \max_{j \in B_i} d_j$,

there exists an optimal schedule in which $B_i \to k$ holds.

Proof. By Corollary 3.1, there must exist an optimal schedule in which $j \to k$ holds for all jobs $j \in B_i$ with $r_j \leq E_k$. Let m ($m \geq 1$) be the number of such job(s), we update E_k so that $E_k \geq r(B_i) + mp$. Suppose that $m < |B_i|$. As B_i is a subblock, there must exist at least one other job $j \in B_i$ with $r_j < r(B_i) + mp \leq E_k$. Again by Corollary 3.1, we have $j \to k$ without loss of optimality. Continue in this way, we have an optimal schedule in which $j \to k$ holds for all $j \in B_i$. □

If a job $k \notin B_i$ with $r_k < t(B_i)$ can be preceded by B_i in an optimal schedule, we have $E_k \geq t(B_i)$. Then, at any decision time within I_i, job k needs no consideration.

Lemma 4.2 *For any subblock B_i and an uncompleted job $k \notin B_i$, if*

(1) only k^+ and the jobs in B_i should be considered within I_i, and

(2) $d_k \geq \max\{t(B_i), \max_{j \in B_i} d_j\}$,

B_i is optimal.

Proof. Consider any p-active schedule S in which some jobs $j \in B_i$ are completed later than k^+ and job l is the one latest completed. As $r_k \leq s_k < r(B_i) \leq r_l$ and $C_k < C_l$, $k^+ \rightarrow l$ holds by Lemma 3.3. Since $p_{k^+} < p$, we have $C_k < t(B_i) < C_l$. We re-schedule k^+ and B_i so that k^+ is postponed as late as possible and completed at C_l, without affecting any other jobs. By assumption (1), all jobs in B_i can be re-scheduled within I_i without postponing the completion of any job $j \in B_i$. If $C_l \leq d_k$, no job's tardiness increases; if $d_k < C_l$, only T_k increases by $\delta T_k = C_l - \max\{d_k, C_k\}$, while T_l decreases by $\delta T_l \geq C_l - \max\{t(B_i), d_l\}$. As $d_k \geq \max\{t(B_i), d_l\}$ by assumption (2), $\delta T_k \leq \delta T_l$, which implies that the re-scheduling should be made. In any case, we get a schedule not worse than S. Thus, there must exist an optimal schedule in which B_i is scheduled within I_i. □

4.2.1 Decomposition in the 1st-stage

Initially, we set U={}. Select a job from B by the latest due date (LDD) rule, which is equivalent to the condition of Equation 3.1 for all due-date-based criterion. Add the selected job to U and decomposition block B by the new set U. Suppose s subblocks are created (ref. Figure 4.1). For the sake of convenience, we let the job in U be j_1 and add the number 1 to the subscripts of the notations B_i, I_i, δ_i, Δ and Λ used in the last chapter. We note that $\Delta_1 < p = \Delta_1 + \Lambda_1$.

Property 4.1 *For $j_1 \in U$,*

(1) $d_{j_1} \geq \max_{j \in B_{1i}} d_j$ holds for all i=1,...,s.

(2) $d_{j_1} \geq r(B_{1s}) \geq t(B_{1i})$ holds for all i=1,...,s-1.

(3) $r_{j_1} < t(B_{11}) \leq r(B_{1i})$ holds for all i=2,...,s.

Proof. (1) According to the LDD rule, $d_{j_1} = \max_{j \in B} d_j \geq \max_{j \in B_{1i}} d_j$ for all i=1,...,s.

(2) Consider any job $l \in B_{1s}$ released at $r(B_{1s})$. As $d_{j_1} \geq \max_{j \in B_{1s}} d_j$, we have $d_{j_1} \geq d_l$. By the initial assumption of $r_i \leq d_i$ for all $i \in N$, $r(B_{1s}) = r_l \leq d_l$. Since $t(B_{11}) \leq r(B_{12}) \leq ... \leq t(B_{1(s-1)}) \leq r(B_{1s})$, $d_{j_1} \geq r(B_{1s}) \geq t(B_{1i})$ holds for all i=1,...,s-1.

(3) $r_{j_1} < t(B_{11})$ is obvious, otherwise, B_{11} and $B \setminus B_{11}$ should be two independent blocks of the overall problem by Definition 3.2. As $t(B_{11}) \leq r(B_{12}) \leq ... \leq r(B_{1s})$, $r_{j_1} < r(B_{1i})$ holds for all i=2,...,s. □

Property 4.2 *All B_{1i}, i=1,...,s-1, must be optimal.*

Proof. Suppose δ_{1l} $(1 \leq l \leq s)$ is the first non-empty interval among all δ_{1i}s. We have $r(B_{11}) = r(B) \leq r_{j_1} = E_{j_1}$. As $d_{j_1} \geq \max_{j \in B_{1i}} d_j$ holds for all i=1,...,s-1 by Property 4.1, there exists an optimal schedule in which $B_{11} \rightarrow j_1$ holds by Lemma 4.1. So, we have $E_{j_1} = r(B_{12})$, and again by Lemma 4.1, we have $B_{12} \rightarrow j_1$ in an optimal schedule. Continue in this way, we have an optimal schedule in which $B_{1i} \rightarrow j_1$ holds for all i=1,...,l-1, which implies that all B_{1i}, i=1,...,l-1, are optimal. As j_1 is the only available job within δ_{1l}, it should be started before but uncompleted until $r(B_{1l})$. So, only the set $B_{1l} \cup \{j_1^+\}$ is available within I_{1l}. As $d_{j_1} \geq \max\{t(B_{1i}), \max_{j \in B_{1i}} d_j\}$ holds for all i=1,...,s-1 by Property 4.1, B_{1l} is optimal by Lemma 4.2. Continue in this way, all B_{1i}, i=l+1,...,s-1, can be proved optimal.□

According to Property 4.2, we can separately schedule all jobs in B_{1i} within I_{1i}, i=1,...,s-1, without loss of optimality, and then, we have the following properties.

Property 4.3 *If $\Delta_1 = 0$, B_{1s} is optimal and j_1 can be an optimal ending job of block B.*

Proof. As $\Delta_1 = 0$, we have $E_{j_1} \geq r(B_{1s})$ now. As $d_{j_1} \geq \max_{j \in B_{1s}} d_j$ by Property 4.1, $B_{1s} \rightarrow j_1$ holds by Lemma 4.1. So, B_{1s} is optimal too, and job j_1 must be wholly scheduled within Λ_1 and completed at $t(B)$. □

According to Property 4.3, there exists an optimal schedule in which job j_1 is scheduled within $[t(B) - p, t(B)]$ if $\Delta_1 = 0$. Generally, for a block in which all jobs have *agreeable* due dates (i.e. $r_i \leq r_j$ implies $d_i \leq d_j$), the job latest released always can be selected first as an optimal ending job and wholly scheduled in the end of the unoccupied interval, since $\Delta_1 = 0$ holds in each of the following decomposition. Finally, we get an optimal schedule without preemption. So, Lemma 4.3 follows Property 4.3.

Lemma 4.3 *The ERD schedule of a block is optimal if the release dates and due dates are agreeable.*

Property 4.4 *If $d_{j_1} \geq t(B_{1s})$, B_{1s} is optimal and j_1 can be an optimal ending job of block B.*

Proof. If $\Delta_1 = 0$, the property holds by Property 3.3. If $\Delta_1 > 0$, j_1 must be uncompleted until $r(B_{1s})$ and only the set $B_{1s} \cup \{j_1^+\}$ is available within I_{1s}. As $d_{j_1} \geq t(B_{1s})$ by assumption and $d_{j_1} \geq \max_{j \in B_{1s}} d_j$ by Property 4.1, B_{1s} is also optimal by Lemma 4.2. Consequently, j_1 is scheduled within $\Delta_1 \cup \Lambda_1$ and completed at $t(B)$. □

Property 4.5 *If $\Delta_1 > 0$ and $d_{j_1} < t(B_{1s})$,*

(1) B_{1s} is not optimal and job j_1 cannot be an optimal ending job of block B.

(2) an optimal ending job of B_{1s} can be an optimal ending job of block B.

Proof. Suppose we continue to schedule B_{1s} with I_{1s}. Job j_1 should be scheduled within $\Delta_1 \cup \Lambda_1$ and completed at $t(B)$. Let j_1^+ denote the piece of job j_1 scheduled within Λ_1. As $\Delta_1 + \Lambda_1 = p$ and $\Delta_1 > 0$, $p_{j_1}^+ = \Lambda_1 < p$. Suppose a job $l \in B_{1s}$ is the one completed at $t(B_{1s})$. By Definition 3.2, we have $r(B_{1s}) \leq r_l$. Since

$\Delta_1 > 0$, $r_{j_1} < r(B_{1_s})$. So, we have $r_{j_1} < r(B_{1_s}) \leq r_l$. We can re-schedule job l and j_1^+, without affecting any other jobs, so that job l is not started before the completion of job j_1. Let C'_{j_1} and C'_l be the completion time of job j_1 and job l in the new schedule, respectively. We have $C'_{j_1} < C_l = t(B_{1_s})$ and $C'_l = C_{j_1} = t(B)$. As $d_l \leq d_{j_1}$ by Property 4.1 and $d_{j_1} < t(B_{1_s})$ by assumption, the total tardiness decreases by $t(B_{1_s}) - \max\{d_{j_1}, C'_{j_1}\} > 0$. So, the new schedule is better than the old one, which means that B_{1_s} is not optimal and job j_1 cannot be completed at $t(B)$ in any optimal schedule. Consequently, a job $l \in B_{1_s}$ must be completed at $t(B)$ in an optimal schedule. □

By Properties 3.2-3.5, in case (1): $\Delta_1 = 0$, or (2): $d_{j_1} \geq t(B_{1_s})$, all the s subblocks are optimal, and the decomposition result is optimal; in case (3): $\Delta_1 > 0$ and $d_{j_1} < t(B_{1_s})$, the result is not optimal, since B_{1_s} is not optimal. Then, we should decompose block B by another subset. In this case, the decomposition for block B is said in *multi-stage*. In what follows, we discuss the decompositions in the uth-stage, $u \geq 2$.

4.2.2 Decomposition in the uth-stage

In case (3) in the 1st-stage, a decomposition for block B by another subset is necessary. We select a new job from B_{1_s} by the LDD rule and add it to U. As a rule, we always let the job added to U in the uth-stage be j_u. Suppose we decompose block B by $U = \{j_1, j_2\}$ in two steps: (1) remove job j_1 and re-schedule the set $B \setminus \{j_1\}$ by the ERD rule, and (2) remove job j_2 and re-schedule the set $B \setminus U$ by the ERD rule. After Step (1), we get an interim result which is identical to that in the 1st-stage (see Figure 4.1). As $j_2 \in B_{1_s}$ and $r(B_{1_s}) \leq r_{j_2}$, Step (2) does not change the existing result before $r(B_{1_s})$. So, following the result in the 1st-stage, we gain the result in the 2nd-stage directly by removing job j_2 from B_{1_s} and re-scheduling the set $B_{1_s} \setminus \{j_2\}$ by the ERD rule. Obviously, the 2ndstage decomposition for block B is equivalent to a decomposition for B_{1s} by job j_2. Suppose the set $B_{1s} \setminus \{j_2\}$ decomposes into z subblocks. The final result is shown in Figure 4.1. Generally, we use B_{ki} to denote the new subblocks created in the kth-stage, and I_{ki} the intervals corresponding to B_{ki}. We also use δ_{ki} to denote the idle interval immediately preceding I_{ki}. However, we define Δ_k to be the union of the δ_{ki}s but not all δs as defined in the last chapter. On the other hand, Λ_k still denotes the last idle interval. Let $\lambda_k = p - \Delta_k$. We have $\Lambda_k = \sum_{l=1}^{k} \lambda_l$ and $\sum_{l=1}^{k} \Delta_l + \Lambda_k = kp$. We note that $\Delta_k < p$ always holds.

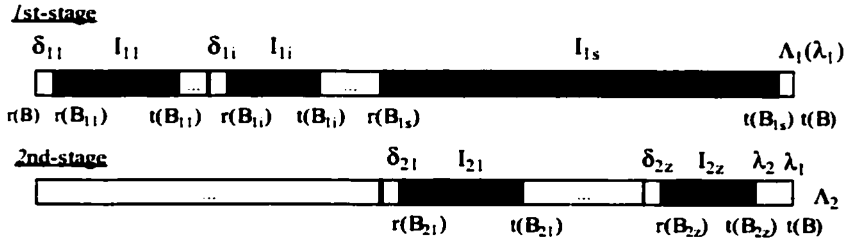

Figure 4.1: The decomposition results in the 1st and 2nd stages

Consider the case of $d_{j_2} < t(B_{2z})$ and $\Delta_2 > 0$, which is similar to case (3) in the 1st-stage. The subblocks B_{1i}, i=1,...,s-1, are obviously optimal, since the available jobs at any decision time within $[r(B), r(B_{1s})]$ are as same as that in the proof of Property 4.2. It will be shown that all the subblocks B_{2i}, i=1,...,z-1, are also optimal and an optimal ending job of B_{2z} can be an optimal ending job of block B, but B_{2z} cannot be proved optimal. Again, the 3rd-stage decomposition for block B is necessary and the result can be directly gained from the result in the 2nd-stage by further decomposition for B_{2z}. Generally, the decomposition for block B can be repeated by decomposing the newest last subblock by a selected job subject to the LDD rule, so long as the further decomposition is necessary, which implies that the newest last subblock cannot be proved optimal. We prove the correctness of the repetition by induction.

Suppose block B is decomposed in u-1, $u \geq 2$, stages and the last subblock in each of the u-1 stages cannot be proved optimal. For the sake of simplicity, we use V to denote the newest last subblock in the (u-1)th-stage. Select a job from V by the LDD rule and add it to U. Decompose block B by $U = \{j_1, ..., j_u\}$. Suppose the set $V \setminus \{j_u\}$ decomposes into v subblocks. Figure 4.2 shows the decomposition results in the (u-1)th and uth stages.

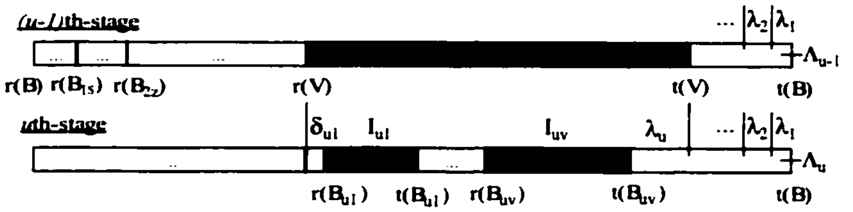
(u-1)th-stage
... λ2 λ1
Λu-1
r(B) r(B1s) r(B2z) r(V) t(V) t(B)
uth-stage
δu1 Iu1 Iuv λu ... λ2 λ1
Λu
r(Bu1) t(Bu1) r(Buv) t(Buv) t(B)

Figure 4.2: The decomposition results in the $(u-1)$th and uth stages

Property 4.6 *For all jobs $j_1, ..., j_u$ in U,*

(1) $d_{j_1} \geq ... \geq d_{j_u} \geq \max_{j \in B_{ui}} d_j$ holds for all i=1,...,v.

(2) $d_{j_1} \geq ... \geq d_{j_u} \geq r(B_{uv}) \geq t(B_{ui})$ holds for all i=1,...,v-1.

(3) $r_{j_1} < ... < r_{j_u} < t(B_{u1}) \leq r(B_{ui})$ holds for all i=2,...,v.

Proof. As $\Delta_k > 0$ in each of the first u-1 stages, $r_{j_1} < ... < r_{j_u}$. According to the LDD rule, $d_{j_1} \geq ... \geq d_{j_u}$. By Property 4.1, we have: (1) $d_{j_u} \geq \max_{j \in B_{ui}} d_j$ for all i=1,...,v; (2) $d_{j_u} \geq r(B_{uv}) \geq t(B_{ui})$ for all i=1,...,v-1; and (3) $r_{j_u} < t(B_{u1}) \leq r(B_{ui})$ for all i=2,...,v. □

Property 4.7 *All the subblocks except B_{uv} must be optimal.*

Proof. (by induction) The correctness in the 1st-stage is proved in Property 4.2. We should prove the correctness in the uth-stage on the assumption of the correctness in the kth-stage for all k=1,...,u-1. By this assumption, *in the result of the (u-1)th-stage, all subblocks except V is optimal.*

As the decomposition for V does not change the existing result before $r(V)$, all subblocks except the new ones decomposed from V are still optimal. So, only the jobs in B_{u1} and some jobs in U are available within I_{u1}. However the jobs in U should be scheduled within the Δs before $r(B_{u1})$, the available jobs from U can be classified into two subsets: (1) the ones not started until $r(B_{u1})$; and (2) the ones started before but uncompleted yet until $r(B_{u1})$. As $d_{j_k} \geq \max_{j \in B_{u1}} d_j$ holds for all $j_k \in U$ by Property 4.6, all jobs in subset (1) can be preceded by B_{u1} in an optimal schedule by Lemma 4.1 and need no consideration within I_{u1}. Suppose we schedule some or all of the uncompleted jobs in subset (2) within I_{u1}, whether partially or wholly, and get a p-active optimal schedule S^*. Let U' be the set of such jobs and j_h^+ be the one latest started. All jobs in U' except j_h^+ must be completed before $t(B_{u1})$, otherwise, S^* cannot be p-active.

Without affecting any other jobs, we re-schedule all jobs in U', including the pieces of these jobs scheduled before $r(B_{u1})$, but excluding the piece(s) of j_h^+ after $t(B_{u1})$, by the ERD rule. As $d_{j_k} \geq t(B_{u1})$ holds for all $j_k \in U$ by Property 4.6, no job's tardiness increases and a new optimal schedule is gained. Let U'' be the set of the jobs from U' that are still scheduled within I_{u1} and job j_g be the leftmost one. Apparently, $r(B_{u1}) < s_{j_k}$ holds for all jobs in $j_k \in U'' \setminus \{j_g\}$. If $r(B_{u1}) \leq s_{j_g}$, All jobs in U'' can be preceded by B_{u1} as the jobs in subset (1) do. Then, we can get another optimal schedule in which B_{u1} is scheduled within I_{u1}. If $s_{j_g} < r(B_{u1})$, we also can get another optimal schedule in which no job in U'' except j_g^+ is scheduled within I_{u1}. As $d_{j_g} \geq \max\{t(B_{u1}), \max_{j \in B_{u1}} d_j\}$ by Property 4.6, B_{u1} is optimal by Lemma 4.2. In both cases, B_{u1} is proved optimal.

Then, only B_{u2} and some jobs in U are available within I_{u2}. Again, however some jobs in U should be scheduled within the Δs before $r(B_{u2})$, the available jobs from U can be classified into two subsets: (1) the ones not started until $r(B_{u2})$: and (2) the ones started before but uncompleted yet until $r(B_{u2})$. We note that the above discussion for the scheduling within I_{u1} is based on no given subsets U' and U''. So, the conclusion also holds for all I_{ui}, i=2,...,v-1, as $d_{j_k} \geq \max\{t(B_{ui}), \max_{j \in B_{ui}} d_j\}$ holds for all $j_k \in U$ and i=1,...,v-1 by Property 4.6. Therefore, all the subblocks B_{ui}, i=1,...,v-1, are optimal. □

Property 4.8 *If $d_{j_u} < t(B_{uv})$ and $\Delta_u > 0$, an optimal ending job of B_{uv} can be an optimal ending job of block B.*

Proof. (by induction) The correctness in the 1st-stage is proved in Property 4.5. We should prove the correctness in the uth-stage on the assumption of the correctness in the kth-stage for all k=1,...,u-1. By this assumption, *an optimal ending job of V can be an optimal ending job of block B.*

Following Property 4.7, we can schedule each of the subblocks except B_{uv} within the corresponding Is without loss of optimality. So, only B_{uv} and some jobs in U are available within I_{uv}. As an optimal ending job of V can be an optimal ending job of block B, there must exist an optimal ending job of block B within the set $B_{uv} \cup \{j_u\}$. Suppose we postpone job j_u as late as possible so that it is completed at $t(B)$, and get an optimal schedule S^*. Let job $l \in B_{uv}$ be the one completed later than any other jobs in B_{uv}. $t(B_{uv}) \leq C_l^*$ is obvious. As $r_{j_u} < r(B_{uv}) \leq r_l$ by Property 4.6, we can re-schedule job j_u and job l without affecting any other jobs, so that job l is completed at $t(B)$. At the same time, job j_u can be completed not later than C_l^*, as $p_{j_u} = p_l = p$. As $d_l \leq d_{j_u}$ by Property 4.6 and $d_{j_u} < t(B_{uv})$ by assumption, the decrease of T_{j_u} is not less than the increase of T_l. So, the total tardiness does not increase and we get a new optimal schedule in which a job $l \in B_{uv}$ is completed at $t(B)$. □

By Properties 4.7 and 4.8, the decomposition for block B can be repeated by decomposing the newest last subblock, so long as the inequalities in Property 4.8 hold in any kth-stage. Apparently, once either of these inequalities is not satisfied, the repetition discontinues. Without loss of generality, we consider the discontinuity at the uth-stage. There are only two cases: (I) $d_{j_u} \geq t(B_{uv}$; and (II)

$d_{j_u} < t(B_{uv})$ but $\Delta_u = 0$. We have the following properties for cases (I) and (II), respectively.

Property 4.9 *If $d_{j_u} \geq t(B_{uv})$, B_{uv} is optimal and job j_u can be an optimal ending job of block B.*

Proof. According to Property 4.7, we can schedule each of the subblocks except B_{uv} within the corresponding Is without loss of optimality. By Property 4.6, $d_{j_k} \geq t(B_{uv})$ and $d_{j_k} \geq \max\limits_{j \in B_{uv}} d_j$ holds for all $j_k \in U$. Following the proof for the optimality of $B_{u(v-1)}$ in the proof of Property 4.7, we can easily prove that B_{uv} is optimal. Consequently, job j_u is completed later than any other jobs in V in an optimal schedule and is an optimal ending job of V. By the assumption that *an optimal ending job of V can be an optimal ending job of block B*, job j_u can be an optimal ending job of block B. □

Property 4.10 *If $d_{j_u} < t(B_{uv})$ and $\Delta_u = 0$, job j_u can be an optimal ending job of block B.*

Proof. According to Property 4.7, we can schedule each of the subblocks except B_{uv} within the corresponding Is without loss of optimality. As $\Delta_u = 0$, we have $E_{j_u} = r(B_{uv})$. As $d_{j_u} \geq \max\limits_{j \in B_{uv}} d_j$ by Property 4.6, there exists an optimal schedule in which $B_{uv} \rightarrow j_u$ holds by Lemma 4.1. So, job j_u can be an optimal ending job of V. By the assumption that *an optimal ending job of V can be an optimal ending job of block B*, job j_u can be an optimal ending job of block B. □

4.2.3 Scheduling of the set U

We consider case (I), i.e. $d_{j_u} \geq t(B_{uv})$ first. By Properties 4.7 and 4.9, all the subblocks are proved optimal and can be separately considered in the next iteration. So, the decomposition is ended at the uth-stage and the jobs in U should be scheduled within all Δs and Λ_u. We consider two cases: (1) $\Lambda_u \leq p$, and (2) $\Lambda_u > p$.

In case (1), the interval $[t(B_{uv}), t(B)]$ could be completely occupied by job j_u, which could be preceded by all other jobs in U as $r_{j_k} < r_{j_u}$ holds for all $j_k \in U \setminus \{j_u\}$ by Property 4.6. Then, no job in $U \setminus \{j_u\}$ would be completed later than $r(B_{uv})$. As $d_{j_u} \geq t(B_{uv}) > r(B_{uv})$ and $d_{j_k} \geq d_{j_u}$ for all $j_k \in U \setminus \{j_u\}$ by Property 4.6, no job in $U \setminus \{j_u\}$ would be tardy. Thus, we can schedule all jobs in U except job j_u within all Δs including Δ_u by the ERD rule, and then, job j_u within the rest part of the Δ_u and $[t(B_{uv}), t(B)]$. We should note that, unlike in the initial ERD schedule of block B, preemptions exist in the ERD schedule of the set U.

In case (2), job j_u should be wholly scheduled within $[t(B) - p, t(B)]$. Remove job j_u from U. We have $|U| = u - 1$. Let $L = t(B) - p$. The reduced set U should be scheduled within all Δs and $[t(B_{uv}), L]$. Suppose we schedule each job $j_k \in U$ within Δ_k, respectively. As $0 < \Delta_k < p$ for all $j_k \in U$, no job in U can be completely scheduled. Using j_k^+ to denote the uncompleted part of j_k at $r(V)$, we have $p_{j_k^+} = \lambda_k$. Scheduling all j_k^+, k=1,...,u-1, within Δ_u and $[t(B_{uv}), L]$ in the reverse order of r_{j_k}, we get a schedule S' for the reduced block $B' = B \setminus \{j_u\}$ (see Figure 4.3).

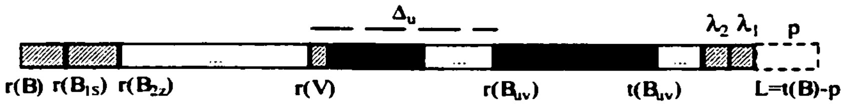

Figure 4.3: A schedule for block B without job j_u

Suppose we decompose B' by job j_1. If $L - \Lambda_1 \geq t(B_{uv})$, the ending time of the last subblock is $L - \Lambda_1$. As job j_1 is selected from B by the LDD rule and $B' \subset B$, it can be an optimal ending job of B' if $d_{j_1} \geq L - \Lambda_1$, and cannot be if $d_{j_1} < L - \Lambda_1$, according to Properties 4.1-4.5. On the other hand, if $L - \Lambda_1 < t(B_{uv})$, which implies that all j_k^+s except a part of j_1^+ are scheduled within Δ_u and only the rest of j_1^+ is scheduled within $[t(B_{uv}), L]$ in S', the ending time of the last subblock is $t(B_{uv})$ rather than $L - \Lambda_1$. As $d_{j_1} \geq d_{j_u} \geq t(B_{uv})$, job j_1 must be an optimal ending job of B' by Property 4.1-4.4. We note that an optimal ending job of B' is also an optimal ending job of U. Generally, we have the following lemma.

Lemma 4.4 *Suppose $d_{j_l} < L - \Lambda_l$ holds for all l=1,...,k-1. Job $j_k \in U$ can be an optimal ending job of U, if*

(1) $d_{j_k} \geq L - \Lambda_k > t(B_{uv})$, or

(2) $L - \Lambda_k \leq t(B_{uv})$.

Proof. As $d_{j_u} \geq t(B_{uv})$ and $d_{j_1} \geq ... \geq d_{j_{u-1}} \geq d_{j_u}$ by Property 4.6, $L - \Lambda_l > d_{j_l} \geq t(B_{uv})$ holds for all l=1,...,k-1, by assumption (1). Following Properties 4.7-4.8, it can be easily verified that, by decomposing B' in k-1 stages, the decomposition for B' can continue without loss of optimality. In case (1), job j_k can be an optimal ending job of B' by Property 4.9. In case (2), the ending time of the last subblock in the kth-stage decomposition for B' is $t(B_{uv})$ rather than $L - \lambda_k$. As $d_{j_k} \geq d_{j_u} \geq t(B_{uv})$, job j_k must be an optimal ending job of B', also by Property 4.9. In both cases, job j_k is an optimal ending job of B' as well as U. □

If a job j_k, $k < u - 1$, can be identified as an optimal ending job of U by Lemma 4.4, it should be scheduled in the same way as that for job j_u. If none of the conditions in Lemma 4.4 is satisfied by any job j_k, k=1,...,u-2, job j_{u-1} must be an optimal ending job of U, as the ending time of the last subblock in the (u-1)th-stage decomposition for B' (by U) must be $t(B_{uv})$ in any case and $d_{j_{u-1}} \geq d_{j_u} \geq t(B_{uv})$ holds. Therefore, all jobs in U can be successfully scheduled by Lemma 4.4. We note that no decomposition for B' is necessary indeed, since all the data needed in Lemma 4.4 are known from the previous decompositions for block B.

Then, we consider case (II), i.e. $d_{j_u} < t(B_{uv})$ but $\Delta_u = 0$. As $\Delta_u = 0$, job j_u can be wholly scheduled within $[t(B) - p, t(B)]$. Again we remove job j_u from U.

We note that B_{uv} is not proved optimal. However, as $B_{uv} \subset V \subset ... \subset B_{2z} \subset B_{1s}$, the analysis about the decomposition for the reduced block $B' = B \setminus \{j_u\}$ still makes sense. So, Lemma 4.4 can be used to search for an optimal ending of B'. Nevertheless, as $d_{j_u} < t(B_{uv})$ now, it is possible that none of the jobs in U satisfies either of the conditions in Lemma 4.4. In such a case, an optimal ending job of B_{uv} can be an optimal ending job of B' by Properties 4.7-4.8, and so, the decomposition for block B should be resumed by decomposing B_{uv}.

4.3 An $O(n^2)$ algorithm

Algorithm SMPP-ELJTT (SMPP, Equal-Length Jobs and Total Tardiness)

1. Set $H_1=...=H_n=\{\phi\}$, and $i=j=1$.
2. Index all jobs by the ERD rule.
3. Schedule all jobs by the ERD rule, and add all blocks to H_1.
4. Index all blocks/subblocks in H_j as $B_1,...,B_{|H_j|}$ by their start times.
5. If $|B_i|=1$, schedule the only job in B_i within I_i, and goto Step 6; else, schedule B_i by Algorithm BLK-DE, and add all optimal subblocks in the final result to H_{j+1}.
6. If $i \neq |H_j|$, set $i=i+1$, and goto Step 5.
7. If $H_{j+1} \neq \{\}$, set $j=j+1$ and $i=1$, and goto Step 4.
8. Stop.

Algorithm BLK-DE (Block Decomposition)

```
1. Set U={}, V=$B_i$ and L=$t(B_i)$.
2. Set $u$=$|U|$+1, select a job from V by the LDD rule and move it to U as $j_u$.
3. If $V \setminus \{j_u\} \neq \{\}$, schedule $V \setminus \{j_u\}$ by the ERD rule; else, goto Step 4.3.
4. Reset V to be the new last subblock and define all the new subblocks
   except V to be optimal.
    4.1. If $d_{j_u} \geq t(V)$, and goto Step 4.3.
    4.2. If $\Delta_u > 0$, goto Step 2.
    4.3. If $L - t(V) > p$, set $k$=$u$; else, goto Step 6.
5. Schedule job $j_k$ within $[L - p, L]$, remove $j_k$ from U.
    5.1. Re-index the jobs in U by the ERD rule, set L=L-p, $k$=1 and $\Lambda = \lambda_1$.
    5.2. If $d_{j_k} < t(V)$, goto Step 2.
    5.3. If $d_{j_k} < L - \Lambda$, set $k$=$k$+1 and $\Lambda = \Lambda + \lambda_k$, and goto Step 5.2.
    5.4. If $L - \Lambda > t(V)$, goto Step 5.6.
    5.5. If $L - t(V) > p$, goto Step 5; else, goto Step 6.
    5.6. If $\Lambda > p$, goto Step 5; else, schedule $\{j_1, ..., j_k\}$ within
         $\cup_{j=1}^{k} \Delta_j$ and $[L - \Lambda, L]$ by the ERD rule,
         remove $\{j_1, ..., j_k\}$ from U, and goto Step 5.1.
6. If $V \neq \{\}$, define V to be optimal, and goto Step 7.
7. Schedule all jobs in $U \setminus \{j_u\}$ within $\cup_{j=1}^{u} \Delta_j$ by the ERD rule,
   schedule job $j_u$ within the rest of $\Delta_u$ and $[t(V), L]$,
   and return to *Algorithm SMPP-ELJTT*.
```

In Algorithm SMPP-ELJTT, H_j, j=1,...,n, is a set of blocks/subblocks that need to be scheduled. Apparently, the optimality of Algorithm SMPP-ELJTT depends on that of Algorithm BLK-DE. To justify Algorithm BLK-DE, we should prove that: (1) the final decomposition result is optimal, and (2) the set of U is optimally scheduled. In Algorithm BLK-DE, Steps 2-4 deal with the multiple stage decomposition for B_i, while Steps 5-7 involve the scheduling of U. The decomposition starts with $|V| > 1$, so $V \setminus \{j_u\} \neq \{\}$ in the 1st-stage. If $|V|=1$ after the decomposition in the uth-stage and the decomposition in the $(u+1)$th-stage is necessary, the only job in V must be an optimal ending job of B_i by Property 4.8 and should be scheduled within $[L-p, L]$.

In Step 4, in the case of (I) $d_{j_u} \geq t(V)$, or (II) $d_{j_u} < t(V)$ but $\Delta_u = 0$, job j_u is an optimal ending job of B_i by Properties 4.9 and 4.10, respectively; otherwise, the decomposition turns to the $(u+1)$th stage. In Step 4.3, according to the analysis in Section 4.3, if $L - t(V) \leq p$, the last idle interval should be completely occupied by job j_u and the scheduling of U ends by Step 7; if $L - t(V) > p$, job j_u should be wholly scheduled within $[L-p, L]$, and the scheduling for U starts.

In Step 5, the conditions in Lemma 4.4 are tested by Steps 5.2-5.3. In case (I), the test always ends with an optimal ending job of U identified. In case (II), $d_{j_k} < t(V)$ means that no job in $\{j_1, ..., j_k\}$ is proved an optimal ending job of U, nor could the jobs in $\{j_{k+1}, ..., j_u\}$ be, as $d_{j_1} \geq ... \geq d_{j_u}$ by Property 4.6. So, further test is unnecessary and the decomposition for B_i is resumed by decomposing V. The scheduling of the identified optimal ending job j_k as well as the rest jobs in U are considered in Steps 5.4-5.6. In Step 5.5, $L - t(V) > p$ implies $\Lambda > p$, as $L - \Lambda \leq t(V)$; in Step 5.6, $\Lambda > p$ implies $L - t(V) > p$, as $L - \Lambda > t(V)$; in both cases, job j_k could be wholly scheduled within $[L-p, L]$ and the test in Steps 5.2-5.3 should be repeated for the rest jobs in U. In the case of $\Lambda \leq p$ in Step 5.6, all jobs in $\{j_1, ..., j_{k-1}\}$ should be scheduled within $\cup_{j=1}^{k} \Delta_j$, as $\cup_{j=1}^{k} \Delta_j = kp - \Lambda \geq (k-1)p$. So, only the jobs in $\{j_{k+1}, ..., j_u\}$ should be tested by Steps 5.2-5.3. $L - t(V) \leq p$ in Step 5 implies $\Lambda \leq p$, as $\Lambda \leq L - t(V)$. Then, the interval $[t(V), L]$ could be wholly occupied by job j_k and no other jobs in U would be tardy. Moreover, the last subblock V is proved optimal now in case (II), as all jobs in U are scheduled without $[r(V), t(V)]$. In any case, the algorithm ends by Step 7 and return to

Algorithm SMPP-ELJTT with an optimal final result, while the set U is optimally scheduled.

Theorem 4.1 *$1|r_i, pmtn, p_i = p|\sum T_i$ can be solved in $O(n^2)$ time.*

Proof. In Algorithm BLK-DE, suppose an optimal final result for a block/subblock B_i is obtained in u_i stages. $u_i < |B_i|$ is obvious. In each of the u_i stages, both Steps 2 and 3 can be done in $O(|V|)$ time, as all jobs in V are already scheduled in ERD order before the decomposition for V. So, $|V| \leq |B_i|$ always holds and the time requirement of all the u_i decompositions is in $O(u_i|B_i|)$ time. In Step 5, an iteration of Steps 5.1-5.3 requires $O(|U|)$ time, $|U| \leq u_i$. If the decomposition is never resumed, at most u_i iterations are necessary and the set U can be scheduled in $O(u_i^2)$ time. If the decomposition is once resumed for some times, which implies that some iterations fail to identify an optimal ending job of U, the number of the iterations are still not more than u_i, since each of the failures corresponds to a job that is directly scheduled within $[L-p, L]$ without an iteration (case (II)). As $u_i < |B_i|$, the time requirement of Algorithm BLK-DE is in $O(u_i|B_i|)$ time. We note that all the u_i jobs selected in the u_i stages are optimally scheduled.

In Algorithm SMPP-ELJTT, suppose the iteration of Steps 4-7 is ended at $j{=}v$, which implies that $H_j = \{\}$ holds for all $j{=}v{+}1,\dots,n$. Let $h_j = |H_j|$ and $u_j = \sum_{i=1}^{h_j} u_i$. As $u_i \leq u_j$ holds for all B_i in H_j, the total time requirement of all subblocks in H_j is in $O(u_j \sum_{i=1}^{h_j} |B_i|)$ time. Let $u_n = \sum_{j=1}^{v} u_j$. $u_n \leq n$ is obvious. As $\sum_{i=1}^{h_j} |B_i| \leq n$ holds for all $j{=}1,\dots,v$, the total time requirement of all the v iterations of Steps 4-7 is in $O(n \sum_{j=1}^{v} u_j){=}O(n^2)$ time. Since Steps 2 and 3 require $O(n \log n)$ and $O(n)$ time, respectively, the algorithm is in $O(n^2)$ time. □

4.4 Discussion

Although preemption is allowed and assumed to be costless in the preemptive cases, it is not absolutely free in practice and should be avoided if possible. So, it is significant to discuss the number of the preemptions. Consider the ties in the search in Step 2 in Algorithm BLK-DE. If two jobs have identical due dates, either one can be selected and move to U without loss of optimality. However, it is easily seen that the remove of a job earlier released generally creates more preemptions than that of a job later released. Hence, it is beneficial to break the ties by the latest release date (LRD) rule in the search, which means that if more than one jobs have identical due dates, the one latest released should be selected. Such an eligible job can be found out with no additional time requirement. Moreover, as shown in Lemma 4.3, if all jobs have agreeable release dates and due dates, a block B can be scheduled by the LDD+LRD rule in $O(|B|\log|B|)$ time.

Baker *et al.* (1983) indicate that their algorithm generates at most n-1 preemptions. The proof is by induction. Suppose a block B decomposes into b subblocks B_i, i=1,...,b, and the schedule for B contains at most $|B_i|$ preemptions for each B_i. As the selected job k is preempted for at most b times, the total number of preemptions is no more than $\sum_{i=1}^{b}(|B_i|-1)+b = |B|-1$ (Baker 1983). In our algorithm, if a block B decomposes into b subblocks after the multiple stage decomposition, the total number of the preemptions for all the jobs in U in the final schedule is at most b, too. In Algorithm BLK-DE, according to the decomposition rule, no job in U is released within any Δs and Λ. So, no a job $j_k \in U$ is preempted by another job $j_i \in U \setminus \{j_k\}$ in any p-active schedule. For all jobs in U, the preemptions by other jobs in $B \setminus U$ may occur only at $r(B_i)$, i=1,...,b, and the total number is no more than b. Therefore, Algorithm SMPP-ELJTT generates at most $n-1$ preemptions.

4.5 Conclusions

In this Chapter, we have considered the problem of preemptive scheduling equal-length jobs and given release dates on a single machine to minimize total tardiness. We have analyzed some optimal properties for a block with equal-length jobs and presented an $O(n^2)$ algorithm for the overall problem. For future research, it will be interesting to investigate the weighted version of this problem and the counterpart to minimize total weighted completion time.

Chapter 5

Preemptive total tardiness problem with agreeable release dates and due dates

5.1 Introduction

The scheduling model on a single machine to minimize total tardiness has extreme significance in modern manufacturing and service industries, since the ability to deliver customer orders on time has become an important factor to the success of a firm. A considerable volume of papers on this model can be found in the literature, but the study has been focused on the special case with equal release dates, i.e. $1||\sum T_i$. The general case with arbitrary release dates, i.e. $1|r_i|\sum T_i$, which has been classified as strongly *NP*-hard (Lenstra *et al.* 1977), is seldom studied. Recently, Koulamas and Kyparisis (2001) proved that the latter problem remains to be strongly *NP*-hard even in case with agreeable release dates and due dates. The preemptive problem $1|r_i, pmtn|\sum T_i$ is *NP*-hard, too. The *NP*-hardness of this problem can be reduced from the *NP*-hardness of $1||\sum T_i$ (Chu

1992). Nevertheless, to the best of our knowledge, neither a pseudo-polynomial algorithm nor a reduction from a known strongly *NP*-hard problem is provided for this problem. So, it is still open to the sense of *NP*-hardness. As a matter of fact, this problem is rarely touched. In this chapter, we study an restricted version of this problem with agreeable release dates and due dates.

The problem addressed can be formally stated as follows: a set of n jobs $N = \{1, ..., n\}$ has to be processed on a single machine which can perform only one job at a time. Each job i has a release date r_i, a processing time p_i, and a due date d_i (without loss of generality, we assume $r_i \leq d_i$). All r_i, p_i and d_i are integers. The processing times and release dates are arbitrary, while the release dates and due dates are agreeable, in the sense that $r_i \leq r_j$ implies $d_i \leq d_j$. Preemption is allowed. The objective is to schedule the jobs so as to minimize total tardiness.

The rest of the chapter is organized as follows: In Section 5.2, we investigate the *NP*-hardness. A decomposition technique and a pseudo-polynomial algorithm are provided in Sections 5.3 and 5.4, respectively. Special cases are discussed in Section 5.5, with some concluding remarks given in Section 5.6.

5.2 *NP*-hardness

Koulamas and Kyparisis (2001) proved the strong *NP*-hardness of $1|(r_i, d_i)|\bar{\mathrm{T}}$, where $\bar{\mathrm{T}}$ denotes the average tardiness, which is equivalent to $1|(r_i, d_i)|\sum T_i$. The proof is based on the definition of an instance of $1|(r_i, d_i)|\bar{\mathrm{T}}$ from an instance of the well-known strongly *NP*-hard problem of $1|r_i|\sum C_i$ by setting $d_i = r_i$ for all $i \in N$. As $d_i = r_i < C_i$ holds in any feasible schedule, $\sum T_i = \sum(C_i - r_i) = \sum C_i - \sum r_i$ holds. As $\sum r_i$ is a constant, minimizing $\bar{\mathrm{T}}$ in $1|(r_i, d_i)|\bar{\mathrm{T}}$ is equivalent to minimizing $\sum C_i$ in $1|r_i|\sum C_i$. The known complexity status of $1|r_i|\sum C_i$ implies that $1|(r_i, d_i)|\bar{\mathrm{T}}$ is also strongly *NP*-hard. As minimizing $\bar{\mathrm{T}}$ is equivalent to minimizing $\sum T_i$, $1|(r_i, d_i)|\sum T_i$ is strongly *NP*-hard, too. Consequently, $1|(r_i, d_i)|\sum w_i T_i$

is strongly *NP*-hard. Similarly, the strong *NP*-hardness of $1|r_i, pmtn|\sum w_iC_i$ implies that $1|(r_i, d_i), pmtn|\sum w_iT_i$ is strongly *NP*-hard. So, we have the following theorem.

Theorem 5.1 *$1|(r_i, d_i), pmtn|\sum w_iT_i$ is strongly NP-hard.*

But, the complexity status of $1|(r_i, d_i), pmtn|\sum T_i$ cannot be determined by this way, as the corresponding problem $1|r_i, pmtn|\sum C_i$ is polynomially solvable. However, the *NP*-hardness of $1|(r_i, d_i), pmtn|\sum T_i$ is obvious, since one of its special cases with $r_i = 0$ for all $i \in N$, where preemption is never necessary, is equivalent to $1||\sum T_i$, a well-known *NP*-hard problem (Lawler 1977, Du and Leung 1990). Thus, we have the following lemma.

Lemma 5.1 *$1|(r_i, pmtn)|\sum T_i$ is NP-hard.*

Nevertheless, as $1||\sum T_i$ is *NP*-hard in the ordinary sense, the sense of *NP*-hardness of $1|(r_i, d_i), pmtn|\sum T_i$ is still undetermined.

5.3 Decomposition technique

We first give some necessary notations as follows.

For all i=1,....,n.

- $P_i = \sum_{j=1}^{i} p_j$;
- $N_i = \{1, 2, ..., i\}$.

In any *p*-active schedule,

- U_k: the set of all jobs with $s_j < s_k$, $j \in N$;

- V_k: the set of all jobs with $s_j \geq s_k$, $j \in N$.

By the above definition, we have $k \in V_k$. Assume all jobs $i \in N$ are indexed and scheduled by the ERD rule (with ties broken by the EDD rule), i.e., $i < j$ implies: (1) $r_i < r_j$; or (2) $r_i = r_j$ and $d_i \leq d_j$. Without loss of generality, we also assume no decomposition occurs in the initial ERD schedule, which implies:

$$P_i > (r_{i+1} - r_1) \tag{5.1}$$

holds for all $1 \leq i \leq n-1$. Obviously, $C_{max} = r_1 + P_n$ holds for any p-active schedule, where C_{max} is the makespan. Let job k be the one having the largest index among all jobs satisfying

$$p_i = \max\{p_j\},\ j \in N \tag{5.2}$$

We consider the case of $k = 1$ and $r_1 < r_2$ first. As J_1 is now the only available job within $[r_1, r_2]$, it should be started at r_1 but cannot be completed at r_2 by Assumption (5.1). Without loss of optimality, we purposely split J_1 into two independent jobs with processing times of $r_2 - r_1$ and $p_1 - (r_2 - r_1)$, respectively. We define the former one J_0 with $r_0 = r_1$, $p_0 = r_2 - r_1$ and $d_0 = r_2$. In the mean time, we define the latter one J_1' with $r_1' = r_2$, $p_1' = r_1 + p_1 - r_2$ and $d_1' = d_1$. The scheduling of the new set $\{J_0\} \cup \{J_1'\} \cup N \setminus \{J_1\}$ is equivalent to the original problem. As J_0 should be scheduled within $[r_2, r_1]$ in any p-active schedule, the rest task is to schedule the set $N' = \{J_1'\} \cup N \setminus \{J_1\}$ within $[r_2, C_{max}]$.

Then, we consider all the other cases, i.e. $1 < k \leq n$ or $k = 1$ but $r_1 = r_2$. According to Theorem 3.1, there exists an optimal schedule in which $j \rightarrow k$ holds, which implies that:

$$s_j < C_j \leq s_k < C_k \tag{5.3}$$

holds for all jobs j=1,2,...,$k-1$. On the other hand, if $s_j < s_k$ holds for a job $j \in \{k+1, ..., n\}$ in a p-active schedule, $j \rightarrow k$ also holds, as $r_k \leq r_j$ now and only

$j \to k$ holds in case of $s_j < s_k$, according to Lemmas 3.2 and 3.3. So, Inequality (5.3) holds for all jobs $j \in U_k$. Define $t(U_k)$ to be the completion time of the latest job $j \in U_k$ in S. Let $P(U_k) = \sum_{j \in U_k}\{p_j\}$, we have $t(U_k) = r_1 + P(U_k)$, which indicates that a p-active schedule S is actually decomposed into two subblocks: U_k and V_k, $V_k = N \setminus U_k$.

Now, we consider the decomposition of any p-active schedule. Our decomposition starts with the initial set of U_k with $U_k = N_{k-1}$ and $t(U_k) = r_1 + P_{k-1}$.

If $t(U_k) < r_{k+1}$, job k should start at $t(U_k)$, since it is the only available job now. The decomposition is finished and we obtain two adjoining subblocks: $U_k = N_{k-1}$ and $V_k = N \setminus N_{k-1}$.

If $t(U_k) \geq r_{k+1}$, we have $r_{k+1} \leq t(U_k) \leq s_k$. According to Lemmas 3.2 and 3.3, either $k \to k+1$ or $k+1 \to k$ holds.

In the case of $k \to k+1$, $C_k \leq s_{k+1}$ always holds. Besides, $C_k \leq d_{k+1}$ must hold. Otherwise, we always can interchange the job pair $k, k+1$ without affecting any other job, while the total tardiness will decrease after the interchanging, as both $d_k \leq d_{k+1} < C_k$ and $p_{k+1} < p_k$ hold. Moreover, suppose another job $j \in \{k+2, ..., n\}$ precedes job k in an optimal schedule which satisfies both $k \to k+1$ and $C_k \leq d_{k+1}$. We have $C_j \leq s_k < C_k \leq d_{k+1}$. As $d_k \leq d_{k+1} \leq d_j$ by Equation (5.2), we always can interchange the job pair j, k without increasing the total tardiness. By as much as necessary times interchanging, we can get an optimal schedule in which $k \to j$ holds for all $j \in \{k+1, ..., n\}$. Again, we obtain two adjoining subblocks: $U_k = N_{k-1}$ and $V_k = N \setminus N_{k-1}$.

In the case of $k+1 \to k$, U_k is extended to be $U_k = N_{k-1} \cup \{k+1\} = N_{k+1} \setminus \{k\}$ with $t(U_k) = r_1 + P_{k+1} - p_k$. Our decomposition continues with the comparison of the new value of $t(U_k)$ with r_{k+2}. Apparently, further decomposition is unnecessary if $t(U_k) < r_{k+2}$. So, the decomposition will either end with two adjoining subblocks: $U_k = N_{k+1} \setminus \{k\}$ and $V_k = N \setminus N_{k+1} \cup \{k\}$, or continues with a new comparison, and so on.

In conclusion, suppose l, $0 \le l \le n-k-1$, to be the minimum integer satisfying $r_1 + P_{k+l} - p_k < r_{k+l+1}$, the decomposition ends when $U_k = N_{k+l} \setminus \{k\}$. In the case that no such an integer exists, the decomposition ends when U_k is updated to be $U_k = N \setminus k$, and in this case, we set l to be $l = n - k$. In any case, we always obtain $l+1$ different decomposition results, each of which consists of two adjoining subblocks. So, we summarize the results in the following theorem.

Theorem 5.2 *Suppose the jobs are indexed by the ERD rule (with ties broken by the EDD+SPT rule) and* $P_i > (r_{i+1} - r_1)$ *holds for all* $1 \le i \le n$. *Let job* k *(*$k \ne 1$ *or* $k = 1$ *but* $r_1 = r_2$*) be the one having the largest index among all jobs satisfying* $p_i = \max\{p_j\}$, $j \in N$, *and* l *be the minimum non-negative integer satisfying* $r_1 + P_{k+l} - p_k < r_{k+l+1}$ *(if no such integer exists, let* $l = n - k$*), then there is some integer* δ, $0 \le \delta \le l$, *such that there exists an optimal schedule composed of two adjoining subblocks:* $U_k = N_{k+\delta} \setminus \{k\}$ *and* $V_k = N \setminus N_{k+\delta} \cup \{k\}$ *for all* $\delta = 0, 1, \ldots l$, *which should be independently scheduled within* $[r_1, t(U_k)]$ *and* $[t(U_k), C_{max}]$, *respectively.*

5.4 A pseudo-polynomial algorithm

Our algorithm is based on dynamic programming. The initial decomposition of the schedule is completed by scheduling all jobs in the ERD order (with ties broken by the EDD+SPT rule). All blocks in the initial ERD schedule have the same properties and can be solved by the same algorithm. The total tardiness of the problem is the sum of the total tardiness of all blocks.

Algorithm SMPP-ARDTT (SMPP, Agreeable Release dates and Due dates, Total Tardiness)

Without loss of generality, we apply our algorithm to a block composed of m jobs. Our objective is to find an optimal schedule beginning at $r(B) = r_1$ and ending at $C_{max} = t(B) = r_1 + P(B)$, $P(B) = P_m$. Let job k be the one having the largest index among all jobs satisfying $p_i = \max\{p_j\}$, $j \in B$, and l be the minimum non-negative integer satisfying $r_1 + P_{k+l} - p_k < r_{k+l+1}$ (if no such integer exists, let $l = n - k$). It follows from Theorem 5.2 that, for some integer δ, $0 \leq \delta \leq l$, there exists an optimal schedule in the form of:

(1) the subblock $U_k = N_{k+\delta} \setminus \{k\}$, starting at $r(B)$ and ending at $r(B) + P_{k+\delta} - p_k$, immediately followed by

(2) the subblock $V_k = N \setminus N_{k+\delta} \cup \{k\}$, starting at $r(B) + P_{k+\delta} - p_k$ and ending at $t(B)$.

The recursion of the dynamic programming is similar to that applied in Lawler (1977). However, for the sake of simplicity and efficiency, the following procedures are necessary:

(1) whether the set that is entering into the recursion is the whole block or a subblock, re-index it as $\{1, 2, ...\}$, by the ERD rule (with ties broken by the EDD+SPT rule), and split J_1, if it is necessary;

(2) for $\delta = l$ (in the case of $0 \leq l < n - k$), split the new J_1, i.e. job k before the re-indexing, and redefine the release dates of all jobs j, $j \in B$, whose actual release dates are earlier than $r(B)$, to $r_j = r(B)$.

All the above procedures are in linear time and do not change the power of the time complexity, so the total time requirement for the block with m jobs is in $O(m^4 P_m)$ time, and consequently, the overall time requirement of the whole problem is in $O(n^4 P)$ time. Thus, we have the following Theorem.

Theorem 5.3 *$1|(r_i, d_i), pmtn|\sum T_i$ is NP-hard in the ordinary sense and can be solved in $O(n^4P)$ time.*

5.5 Special cases

5.5.1 Special case I: $r_i = r$

In this case, $r_i = r_j = r$ (r is a constant) holds for any job pair $i, j \in N$, equivalently, all jobs are released simultaneously. As indicated in Section 5.2, this problem is equivalent to $1||\sum T_i$, which is ordinary *NP*-hard and can be pseudo-polynomially solved (Du and Leung 1990 and Lawler 1977).

5.5.2 Special case II: $d_i = d$

In this case, $d_i = d_j = d$ (d is a constant) holds for any arbitrary job pair $i, j \in N$ and the problem can be denoted as $1|r_i, d_i = d, pmtn|\sum T_i$. It can be easily reduced from Theorem 3.1 that this problem can be solved by the SRPT rule.

Lemma 5.2 *For $1|r_i, d_i = d, pmtn|\sum T_i$, the SRPT rule is optimal.*

5.5.3 Special case III: (r_i, d_i, p_i)

In this case, the problem can be denoted as $1|(r_i, d_i, p_i), pmtn|\sum T_i$. According to Theorem 3.1, there exists an optimal schedule $i \rightarrow j$ holds for any job pair i, j satisfying $r_i < r_j$ and $p_i \leq p_j$, which means that any job should not start before the completion of all the jobs earlier released. So, none preemption is necessary and an optimal schedule is composed with several sub-schedules, each of which consists of the jobs with the same release dates. On the other hand, according to the definitions in Chapter 2, due dates and processing times are agreeable for all jobs with identical release dates. So, each of the sub-schedules can be gained by the EDD rule without preemption. Thus, we have the following lemma.

Lemma 5.3 *For both $1|(r_i, d_i, p_i), pmtn|\sum T_i$ and $1|(r_i, d_i, p_i)|\sum T_i$, the ERD+EDD rule is optimal.*

5.6 Conclusions

We have considered the problem of the single machine preemptive scheduling with agreeable release dates and due dates to minimize total tardiness. We proved this problem to be NP-hard and provided a pseudo-polynomial algorithm for it. So we asserted this problem to be NP-hard in the ordinary sense. Some special cases are identified to be polynomially solvable. Despite our efforts, neither a pseudo-polynomial algorithm nor a reduction from a strongly NP-hard problem is found for the general version of this problem, in which the due dates are not always agreeable with the release dates. So, we would like to mention that the sense of the NP-hardness of $1|r_i, pmtn|\sum T_i$ is still open.

Chapter 6

Due date assignment problems with release dates

6.1 Introduction

After the publication of the classical survey of due date assignment research by Cheng and Gupta (1989), a large amount of literature dealing with the due date assignment problems appears in the last ten years. As indicated by Gordon *et al.* (2002a, 2002b), the new literature has still been focused on the static job shop situation. However, the study on the static models has been widely extended. One of the main directions of the extension is from the single machine environment to the multiple machine environment. A lot of papers on the multiple machine models are mentioned in Gordon's *et al.* two surveys. The latest result can be found in Cheng and Kovalyov (1999). One other main direction is from the simplest CON method to much more sophisticated methods, such as SLK and PPW. Gordon *et al.* (2002b) focused on the due date assignment problems involving such methods. The latest results can be found in Gordon and Strusevich (1999) for the SLK method and in Kahlbacher and Cheng (1995) for the PPW method. Another main direction is from the classical view that due dates are treated as decision variables but are assigned to corresponding jobs before the scheduling, to an alternate view that each due date is not associated with a specific job but assigned to a job according to the sequencing result. The due date assignment method from the alternate view has drawn much attention and has been a hot research topic for a few years. Gordon *et al.* (2002b) also provided an extensive review on this field. The newest result can be found in Qi *et al.* (2001).

There is still another important direction of the extension in the study on the static single machine problem: from the simultaneously arriving model to the intermittently arriving model. In the latter model, not all jobs are released at the same time, but the characteristics of all jobs are known to the scheduler before the scheduling. The latter model is more realistic than the former one in some cases in service industry, where the customers usually arrive intermittently. Apparently, an appropriate delivering period is vital to the reputation and benefit of a servicing business and should be carefully determined. Compared with the exogenous due date assignment methods, the endogenous ones are undoubtedly superior. The most popular endogenous methods are SLK and PPW, in which the due date of a job is based on both of its arriving time and processing time. More sophisticated endogenous methods that consider shop status information have been proposed too, such as JIQ and JIS. However, when release dates are considered, analysis of the endogenous due date assignment methods is so complicated that only a very few papers are concentrated on such models. The latest results can be found in Gordon (1993) and Cheng and Gordon (1994).

In this chapter, we extend Cheng and Gordon's study to the total cost problems with CON/SLK/TWK due dates, but focus on the CON due date assignment method. The problems addressed can be formally stated as follows: a set of n jobs $N = \{1, \dots, n\}$ has to be processed on a single machine which can perform only one job at a time. Each job i has a release date r_i, a processing time p_i, a weight

w_i, and a due date d_i (without loss of generality, we assume $r_i \leq d_i$). All r_i, p_i, w_i and d_i are integers. The processing times and release dates are arbitrary, while the weights are either arbitrary or identical. The due dates are determined by either of the CON, SLK and TWK due date assignment methods. The objective is to schedule the jobs so as to minimize:

(1) maximum tardiness T_{max}, or

(2) weighted number of tardy jobs $\sum(w_i)U_i$, or

(3) total weighted tardiness $\sum(w_i)T_i$.

The rest of the chapter is organized as follows: In Section 6.2, we study the complexity of the problems with CON due dates. SLK and TWK models are discussed in Sections 6.3 and 6.4, respectively, with some concluding remarks given in Section 6.5.

6.2 CON models

It is easy to see that, a problem with $d_i = r_i + d$ is a special case of the counterpart with (r_i, d_i), and the former is not harder than the latter. Similarly, a problem with (r_i, d_i) is a special case of the counterpart with arbitrary r_i, and the former is not harder than the latter also.

6.2.1 Maximum tardiness

According to Gordon (1993), both $1|(\bar{r}_i, d_i), tree|f_{max}$ and $1|r_i, pmtn, tree|f_{max}$ are polynomially solvable in $O(n \log n)$ time. So, both $1|r_i, d_i = r_i + d|T_{max}$ and $1|r_i, d_i = r_i + d, pmtn|T_{max}$ can be solved in $O(n \log n)$ time, too. Specifically, for both of the CON problems, the ERD schedule without preemption is identical with the EDD schedule and is optimal.

Theorem 6.1 *$1|r_i, d_i = r_i + d|T_{max}$ and $1|r_i, d_i = r_i + d, pmtn|T_{max}$ can be solved by the ERD rule in $O(n \log n)$ time.*

6.2.2 Number of tardy jobs

Since both $1|(r_i, d_i)|\sum w_iU_i$ and $1|(r_i, d_i), pmtn|\sum w_iU_i$ can be pseudo-polynomially solved in $O(nW)$ time, and both $1|(r_i, d_i)|\sum U_i$ and $1|(r_i, d_i), pmtn|\sum U_i$ can be polynomially solved in $O(n \log n)$ time (Lawler 1994), we conclude that both $1|r_i, d_i = r_i + d|\sum w_iU_i$ and $1|r_i, d_i = r_i + d, pmtn|\sum w_iU_i$ are pseudo-polynomially solvable in $O(nW)$ time, and both $1|r_i, d_i = r_i + d|\sum U_i$ and $1|r_i, d_i = r_i + d, pmtn|\sum U_i$ are polynomially solvable in $O(n \log n)$ time. For both $1|r_i, d_i = r_i + d|\sum w_iU_i$ and $1|r_i, d_i = r_i + d, pmtn|\sum w_iU_i$, polynomial algorithm does not exist, since the problem $1|d_i = d|\sum w_iU_i$, which is a special case of both $1|r_i, d_i = r_i + d|\sum w_iU_i$ and $1|r_i, d_i = r_i + d, pmtn|\sum w_iU_i$ with $r_i = 0$ for all $i \in N$, is NP-hard in the ordinary sense (Lawler and Moore 1969). Thus, we have the following theorem.

Theorem 6.2 *$1|r_i, d_i = r_i + d|\sum U_i$ and $1|r_i, d_i = r_i + d, pmtn|\sum U_i$ can be solved in $O(n \log n)$ time.*

Theorem 6.3 *$1|r_i, d_i = r_i + d|\sum w_iU_i$ and $1|r_i, d_i = r_i + d, pmtn|\sum w_iU_i$ are NP-hard in the ordinary sense and can be solved in $O(nW)$ time.*

6.2.3 Total tardiness

In Koulamas and Kyparisis (2001), the strong NP-hardness of $1|(r_i, d_i)|\sum T_i$ is proved by a reduction from $1|r_i|\sum C_i$ by setting $d_i = r_i$ for all $i \in N$. In the same way, we proved that $1|(r_i, d_i), pmtn|\sum w_iT_i$ is strongly NP-hard in the last chapter, where we also proved that $1|(r_i, d_i), pmtn|\sum T_i$ is NP-hard in the ordinary sense. Apparently, by setting $d = 0$, we can easily prove the strong *NP*-hardness of all the problems $1|r_i, d_i = r_i + d|\sum T_i$, $1|r_i, d_i = r_i + d|\sum w_iT_i$, and $1|r_i, d_i = r_i + d, pmtn|\sum w_iT_i$. Therefore, we have the following theorem.

Theorem 6.4 *The following problems are strongly NP-hard:*
(1) $1|r_i, d_i = r_i + d|\sum T_i$,
(2) $1|r_i, d_i = r_i + d|\sum w_iT_i$, *and*
(3) $1|r_i, d_i = r_i + d, pmtn|\sum w_iT_i$.

Nevertheless, the problem $1|r_i, d_i = r_i + d, pmtn|\sum T_i$ is open to date. On the one hand, it cannot be proved *NP*-hard by a reduction from $1|r_i, pmtn|\sum C_i$ by setting $d = 0$ for all $i \in N$, as we did for the problems in Theorem 6.4, since $1|r_i, pmtn|\sum C_i$ is polynomially solvable (Baker 1974). On the other hand, it cannot be proved *NP*-hard by setting $r_i = 0$ for all $i \in N$, as we did in the last chapter for the problem $1|(r_i, d_i), pmtn|\sum T_i$, since its special case with $r_i = 0$ for all $i \in N$ corresponds to the problem $1|d_i = d|\sum T_i$, which is polynomially solvable in $O(n \log n)$ time (Lawler and Moore 1969). Yet, the following property discloses the *NP*-hardness of $1|r_i, d_i = r_i + d, pmtn|\sum T_i$.

Property 6.1 $1|r_i, d_i = r_i + d, pmtn|\sum T_i$ *is at least as hard as* $1||\sum T_i$.

Proof. We construct the following instance of $1|r_i, d_i = r_i + d, pmtn|\sum T_i$ with a job set of $M \cup N$. We assume that all jobs in M and N are indexed by the ERD rule separately.

- number of jobs: $|M|=m \geq 1$, $|N|=n$.
- release dates: $r_j = 0$, $j \in M$; $0 < r_i \leq m$, $i \in N$.
- processing times: $p_j = 1$, $j \in M$; $p_i > 1$, $i \in N$.
- due dates: $d_l = r_l + m$, $l \in M \cup N$.

As all the m unit jobs in M are available at time 0 and have total processing time of m, there exists no idleness in any p-active schedule. Consider an arbitrary p-active schedule S in which a $(1 \leq a \leq m)$ unit intervals within $[0, m]$ are occupied by some jobs from N. Apparently, exactly a jobs from M must be scheduled after time m. As $d_j = r_j + m = m$ holds for all $j \in M$, all the a jobs are tardy, while the other $m - a$ jobs in M are on time. For each of the a tardy jobs $j \in M$, we can find a corresponding unit interval within $[0, m]$ that is occupied by a unit piece of a job $i \in N$. Since preemption is allowed, we always can interchange a job $j \in M$ with the corresponding unit piece of a job $i \in N$. Let C'_l, $l = i, j$, be the completion time of job l after the interchanging. We have $C'_j \leq m$ and $C'_i = \max\{C_i, C_j\}$. After the interchanging, T_j decreases by $\delta T_j = C_j - m > 0$, as $C_j > m$ and $C'_j \leq m = d_j$; while T_i remains unchanged (in the case of $C_j < C_i$ or $C_i < C_j \leq d_i$) or increases by $\delta T_i = C_j - \max\{C_i, d_i\}$ (in the case of $\max\{C_i, d_i\} < C_j$). As $d_i = r_i + m > m$, $\delta T_j > \delta T_i$ holds in any case. So, the interchanging should be made and the interval $[0, m]$ should be occupied by the m unit jobs $j \in M$ in any optimal schedule. Consequently, all the n jobs in N should be scheduled after time m. The m unit jobs can be scheduled within $[0, m]$ arbitrarily, since no one of them would be tardy. Thus, the whole set $M \cup N$ is optimally scheduled if and only the subset N is optimally scheduled. As all the n jobs in N are available but unscheduled at time m, we can easily transform the scheduling of the subset N to an equivalent problem $1||\sum T_i$ in which each job $i \in N$ has a new due date $d'_i = d_i - m = r_i$. Recall that p_i is independent of r_i for any $i \in N$ by definition. Thus, $1|r_i, d_i = r_i + d, pmtn|\sum T_i$ is at least as hard as $1||\sum T_i$. □

Property 6.1 shows that $1|r_i, d_i = r_i + d, pmtn|\sum T_i$ is NP-hard, since $1||\sum T_i$ is well known NP-hard in the ordinary sense. On the other hand, as $1|r_i, d_i = r_i + d, pmtn|\sum T_i$ is not harder than $1|(r_i, d_i), pmtn|\sum T_i$, which is pseudo-polynomially solvable by the algorithm that we presented in the last chapter, it is pseudo-polynomially solvable, too. Thus, we have the following theorem.

Theorem 6.5 *$1|r_i, d_i = r_i + d, pmtn|\sum T_i$ is NP-hard in the ordinary sense and can be solved in $O(n^4P)$ time.*

6.2.4 Complexity results

We have investigated the complexity of all the regular models with CON due dates. The results are summarized in Table 6.1.

Table 6.1: Complexity results of CON due date assignment models

Problem	Status	Reference
$1\|r_i, d_i = r_i + d\|T_{max}$	$O(n \log n)$	Gordon (1993)
$1\|r_i, d_i = r_i + d, pmtn\|T_{max}$	$O(n \log n)$	Gordon (1993)
$1\|r_i, d_i = r_i + d\|\sum U_i$	$O(n \log n)$	Lawler (1994)
$1\|r_i, d_i = r_i + d, pmtn\|\sum U_i$	$O(n \log n)$	Lawler (1994)
$1\|r_i, d_i = r_i + d\|\sum w_iU_i$	o-*NP*, $O(nW)$	Theorem 6.3
$1\|r_i, d_i = r_i + d, pmtn\|\sum w_iU_i$	o-*NP*, $O(nW)$	Theorem 6.3
$1\|r_i, d_i = r_i + d\|\sum T_i$	s-*NP*	Theorem 6.4
$1\|r_i, d_i = r_i + d, pmtn\|\sum T_i$	o-*NP*, $O(n^4P)$	Theorem 6.5
$1\|r_i, d_i = r_i + d\|\sum w_iT_i$	s-*NP*	Theorem 6.4
$1\|r_i, d_i = r_i + d, pmtn\|\sum w_iT_i$	s-*NP*	Theorem 6.4

6.3 SLK models

First of all, it is easy to verify that, by SLK due date assignment methods, d_i is unnecessarily agreeable with r_i. So, the pseudo-polynomially or polynomially solvable problems in Table 6.1 whose solvability is based on the precondition of agreeable release dates and due dates can no longer be solved by the corresponding algorithms. Nevertheless, all the preemptive problems except the one for minimizing total tardiness still are solvable by the algorithms for the corresponding problems with no restrictions on release dates and due dates. In particular, we have the following theorem.

Theorem 6.6
(1) $1|r_i, d_i = r_i + p_i + d, pmtn|T_{max}$ *can be solved in* $O(n \log n)$ *time.*
(2) $1|r_i, d_i = r_i + p_i + d, pmtn|\sum U_i$ *can be solved in* $O(n^4)$ *time.*
(3) $1|r_i, d_i = r_i + p_i + d, pmtn|\sum w_i U_i$ *can be solved in* $O(n^3 W^2)$ *time.*

Next, consider the strongly *NP*-hard cases in Table 6.1. Setting $d = 0$, we have $d_i = r_i + p_i \le C_i$. As $\sum T_i = \sum(C_i - d_i) = \sum C_i - \sum r_i - \sum p_i$ and both $\sum r_i$ and $\sum p_i$ are constants, minimizing $\sum(w_i)T_i$ is still equivalent to minimizing $\sum(w_i)C_i$. So, we have the following theorem.

Theorem 6.7 *The following problems are strongly NP-hard:*
(1) $1|r_i, d_i = r_i + p_i + d|\sum T_i$.
(2) $1|r_i, d_i = r_i + p_i + d|\sum w_i T_i$.
(3) $1|r_i, d_i = r_i + p_i + d, pmtn|\sum w_i T_i$.

Finally, consider the instance in Property 6.1. Let's redefine the due dates to be $d_l = r_l + p_l + (m - 1)$, $l \in M \cup N$. We also can transform the scheduling of the subset N to an equivalent problem $1||\sum T_i$ in which each job $i \in N$ has a new due date $d'_i = d_i - m = r_i + p_i - 1$. So, $1|r_i, d_i = r_i + p_i + d, pmtn|\sum T_i$ is at least as hard as $1||\sum T_i$. Since $1||\sum T_i$ is *NP*-hard in the ordinary sense, we have the following lemma.

Lemma 6.1 $1|r_i, d_i = r_i + p_i + d, pmtn|\sum T_i$ *is* NP*-hard.*

The complexity results of the SLK due date assignment models are summarized in Table 6.2.

Table 6.2: Complexity results of SLK due date assignment models

Problem	Status	Reference
$1\|r_i, d_i = r_i + p_i + d\|T_{max}$	**open**	
$1\|r_i, d_i = r_i + p_i + d, pmtn\|T_{max}$	$O(n \log n)$	Gordon (1993)
$1\|r_i, d_i = r_i + p_i + d\|\sum U_i$	**open**	
$1\|r_i, d_i = r_i + p_i + d\|\sum w_iU_i$	**open**	
$1\|r_i, d_i = r_i + p_i + d, pmtn\|\sum U_i$	$O(n^4)$	Baptiste (1999a)
$1\|r_i, d_i = r_i + p_i + d, pmtn\|\sum w_iU_i$	o-*NP*, $O(n^3W^2)$	Lawler (1990)
$1\|r_i, d_i = r_i + p_i + d\|\sum T_i$	s-*NP*	Theorem 6.7
$1\|r_i, d_i = r_i + p_i + d\|\sum w_iT_i$	s-*NP*	Theorem 6.7
$1\|r_i, d_i = r_i + p_i + d, pmtn\|\sum T_i$	*NP*, **open**	Lemma 6.1
$1\|r_i, d_i = r_i + p_i + d, pmtn\|\sum w_iT_i$	s-*NP*	Theorem 6.7

6.4 TWK models

As d_i is also unnecessarily agreeable with r_i by the TWK due date assignment method, we have the following theorem.

Theorem 6.8

(1) $1|r_i, d_i = r_i + kp_i, pmtn|T_{max}$ *can be solved in* $O(n \log n)$ *time.*

(2) $1|r_i, d_i = r_i + kp_i, pmtn|\sum U_i$ *can be solved in* $O(n^4)$ *time.*

(3) $1|r_i, d_i = r_i + kp_i, pmtn|\sum w_iU_i$ *can be solved in* $O(n^3W^2)$ *time.*

For the strongly *NP*-hard cases in Table 6.1, we also have $d_i = r_i + p_i \le C_i$ by Setting $k = 1$. Thus, we have the following theorem.

Theorem 6.9 *The following problems are strongly NP-hard:*

(1) $1|r_i, d_i = r_i + kp_i|\sum T_i$.

(2) $1|r_i, d_i = r_i + kp_i|\sum w_iT_i$.

(3) $1|r_i, d_i = r_i + kp_i, pmtn|\sum w_iT_i$.

The complexity results of the TWK due date assignment models are summarized in Table 6.3.

Table 6.3: Complexity results of TWK due date assignment models

Problem	Status	Reference
$1\|r_i, d_i = r_i + kp_i\|T_{max}$	**open**	
$1\|r_i, d_i = r_i + kp_i, pmtn\|T_{max}$	$O(n\log n)$	Gordon (1993)
$1\|r_i, d_i = r_i + kp_i\|\sum U_i$	**open**	
$1\|r_i, d_i = r_i + kp_i\|\sum w_iU_i$	**open**	
$1\|r_i, d_i = r_i + kp_i, pmtn\|\sum U_i$	$O(n^4)$	Baptiste (1999a)
$1\|r_i, d_i = r_i + kp_i, pmtn\|\sum w_iU_i$	o-*NP*, $O(n^3W^2)$	Lawler (1990)
$1\|r_i, d_i = r_i + kp_i\|\sum T_i$	s-*NP*	Theorem 6.9
$1\|r_i, d_i = r_i + kp_i\|\sum w_iT_i$	s-*NP*	Theorem 6.9
$1\|r_i, d_i = r_i + kp_i, pmtn\|\sum T_i$	**open**	
$1\|r_i, d_i = r_i + kp_i, pmtn\|\sum w_iT_i$	s-*NP*	Theorem 6.9

6.5 Conclusions

We have considered the due date assignment problems on a single machine with release dates. In each of the problems, n jobs with given release dates should be scheduled on a single machine, while the due dates are determined by the CON/SLK/TWK method. All models are concerned with minimizing a regular criterion. With a few exceptions the complexity of most of the problems are explicitly determined. For further study, all the open problems in Tables 6.2 and 6.3 are worthy of examination. Besides, the multiple machine versions of all types of due date assignment models with given release dates are a broad research area. Also, the earliness and tardiness models with given release dates are under-explored.

Chapter 7

Total tardiness problem without release dates

7.1 Introduction

Among the classical scheduling problems, the single machine total tardiness problem without release dates ($1||\sum T_i$) is one of the most widely researched. From here onward, we use SMTTP to denote this problem. Since the first theoretical development by Emmons (1969), many papers focused on SMTTP have been published. Two papers in the literature are of important significance in the study on this famous problem. Lawler (1977) gave a pseudo-polynomial algorithm to solve SMTTP in $O(n^4P)$ time, which implies that this problem cannot be strongly *NP*-hard. By a reduction from a restricted version of the *NP*-hard Even-Odd Partition problem, Du and Leung (1990) uncovered the complexity status of SMTTP, which remained open for decades. Du and Leung's results, together with Lawler's pseudo-polynomial algorithm, clearly indicated the ordinary *NP*-hardness of SMTTP. Before the publication of Du and Leung (1990), the majority part of the literature was concentrated on finding a polynomial algorithm or a reduction proving its *NP*-

hardness. So, a wide variety of enumerative algorithms were proposed. Most of the algorithms rely heavily on the dominance rules developed by Emmons (1969) and Lawler (1977), both of which are extended by other researchers. Following the disclosure of the complexity status of this problem, a substantial body of the literature was centered on the heuristic algorithms. Koulamas (1994) gave a comprehensive review on total tardiness problems with an emphasis on critically evaluating heuristic algorithms for SMTTP. However, as indicated by Chen *et al.* (1998), developing approximation algorithms with good performance guarantees for this problem is difficult. The best ratio guarantee for any of the proposed heuristics is $n/2$. On the other hand, Szwarc and some other researchers are still focusing on the development of the exact algorithms, for which both dynamic programming and branch and bound, as well as their hybrid, are applied. Koulamas (1994) also gave a brief survey of the enumerative algorithms for SMTTP. More detailed review on this aspect was presented by Chen *et al.* (1998). The most recent result can be found in Szwarc *et al.* (2001), which contains a branch algorithm that handles instance with up to 500 jobs.

Since SMTTP is *NP*-hard in the ordinary sense, it is worthy to investigate the polynomially solvable cases. Unfortunately, the relative results are few. The common due date problem $1|d_i = d|\sum T_i$ can be solved by the SPT rule according to Lawler and Moore (1969), while the equal processing times problem $1|p_i = p|\sum T_i$ is obviously solvable by the EDD rule, since it is identical with the unit processing times problem $1|p_i = 1|\sum T_i$ in the case of identical release dates. In the case that the due dates and processing times are agreeable, the problem $1|(d_i, p_i)|\sum T_i$ is solvable in $O(n \log n)$ time by the SPT or EDD rule according to Theorem 3 of Lawler (1977), where (d_i, p_i) denotes the agreeability of due dates and processing times. Emmons (1969) showed three special cases where SMTTP can be polynomially solved. Lawler (1977) extended two of Emmons' result. Koulamas (1994) reviewed the polynomially solvable cases of SMTTP and developed the other result of Emmons.

In this chapter, we extend the results in the previous chapters for the problem $1|(r_i, d_i), pmtn|\sum T_i$, which takes SMTTP as a special case with all release dates being identical. to SMTTP. We first propose an alternative proof for a famous theorem in Lawler (1977) and identify some optimality properties. Then, a special case of SMTTP with a given number of distinct due dates is proved polynomially solvable. The problem addressed in this chapter can be formally stated as follows: A set of n jobs $N = \{1, ..., n\}$ has to be processed on a single machine that can perform only one job at a time. Each job i has a processing time p_i and a due date d_i. All p_i and d_i are integers. The objective is to schedule the jobs so as to minimize total tardiness.

The rest of this chapter is organized as follows. In Section 7.2, an alternative proof for Theorem 3 of Lawler (1977) is proposed. Optimality properties and complexity analysis for a special case are presented in Section 7.3. In Section 7.4, the results in previous sections are extended to a general case with release dates. Some conclusion remarks are given in Section 7.5.

7.2 An alternative proof for Theorem 3 of Lawler (1977)

Lawler's (1977) pseudo-polynomial algorithm is based on the key theoretical development of Theorem 3 of Lawler (1977), which is based on two preliminary results (Theorems 1 and 2 of Lawler 1977). The sensitivity of an optimal schedule to the due dates is considered in Theorem 1, while a dominance rule is presented in Theorem 2. Taking the idea of the introduction for Theorem 5.2 in this thesis, we provide an alternative proof for Lawler's Theorem 3. We first recall the former part of Lawler's Theorem 2, which is also a result from Theorem 1 of Emmons (1969). We note that Lawler's theorems are proposed for the problem $1||\sum w_i T_i$

with reversely agreeable weights and processing times, namely, $1|(p_i, -w_i)|\sum w_iT_i$. Apparently, SMTTP is a special case of this weighted problem.

Theorem 7.1 *(ref. Theorem 2 of Lawler 1977 and Theorem 1 of Emmons 1969) Suppose the jobs are agreeably weighted. Then there exists an optimal sequence π in which job i precedes job j if $d_i \le d_j$ and $p_i \le p_j$.*

Theorem 7.2 *(Theorem 3 of Lawler 1977)*
Suppose the jobs are agreeably weighted and numbered in nondecreasing due date order, i.e. $d_1 \le d_2 \le \dots \le d_n$. Let job k be such that $p_k = \max_j\{p_j\}$. There is some integer δ, $0 \le \delta \le n - k$, such that there exists an optimal schedule π in which k is preceded by all jobs j such that $j \le k + \delta$, and followed by all jobs j such that $j > k + \delta$.

Proof. By Theorem 7.1, there exists an optimal schedule S^* in which $j \to k$ holds for all $j < k$. Suppose $i + 1 \to k \to i$ holds for a job pair $i, i + 1$, $k < i < n$. In the case of $C_k^* \le d_i$, $\sum w_iT_i$ does not increase if we postpone job $i+1$ and advancing the following jobs without changing their orders so that job k is immediately followed by job $i + 1$, as the only postponed job (job $i + 1$) is completed at C_k^* after the interchanging and $C_k^* \le d_i \le d_{i+1}$ by assumption. In the case of $C_k^* > d_i$, a simple interchange of the pair k, i does not increase $\sum w_iT_i$, since $p_k = \max_j\{p_j\}$, $w_k \le w_i$ and $d_k \le d_i < C_k^*$ by assumption. In both cases, we get a new optimal schedule in which either $\{i, i + 1\} \to k$ or $k \to \{i, i + 1\}$ holds. Repeatedly applying these two rules, we must obtain an optimal schedule π in which for any job pair $j, j + 1$, $k < j < n$, either $j \to k \to j + 1$, or $\{j, j + 1\} \to k$, or $k \to \{j, j + 1\}$ holds. $\square$

7.3 Optimality properties

Consider a special case of $1||\sum T_i$ in which there are m distinct due dates $d'_1 < \ldots < d'_m$, where $m \ll n$ is a given positive integer. Define N_i to be the subset of the $n_i = |N_i|$ jobs with the common due date d'_i, i=1,...,m. Also, suppose the n_i jobs in N_i are indexed by the SPT rule as $J_{i1}, \ldots, J_{in_i}$. According to Theorem 7.1, there exists an optimal schedule in which $J_{i1} \rightarrow \ldots \rightarrow J_{in_i}$ holds for all i=1,...,m.

Property 7.1 *Let J_{kn_k} be such a job in N_k that $p_{kn_k} = \max_j\{p_{jn_j}\}$. There exists an optimal schedule in which either $J_{kn_k} \rightarrow N_i$ or $N_i \rightarrow J_{kn_k}$ holds for all $i = 1, \ldots, k-1, k+1, \ldots, m$.*

Proof. By assumption, $p_{kn_k} \geq p_{ij}$ for all i=1,...,m and j=1,...,n_i. By Theorem 7.2, there exists an optimal schedule S^* in which J_{kn_k} is preceded by all jobs in N_i, i=1,...,$k-1$. Suppose $J_{ij} \rightarrow J_{kn_k} \rightarrow J_{i(j+1)}$ holds for some $k+1 \leq i \leq m$ and $1 \leq j < n_i$. By assumption, we have $d_k \leq d_i$. In the case of $C^*_{kn_k} \leq d_i$, $\sum T_i$ does not increase if we postpone J_{ij} so that J_{kn_k} is immediately followed by J_{ij}, as the only postponed job (J_{ij}) is completed at $C^*_{kn_k}$ after the interchanging and $C^*_{kn_k} \leq d_i$ by assumption. Similarly, $\sum T_i$ does not increase if we postpone $J_{i(j-1)}$ in the same way. Continue in this way, we can postpone all $J_{ij}, \ldots, J_{i1}$ without increase to $\sum T_i$ and get an optimal schedule in which $J_{kn_k} \rightarrow N_i$ holds. In the case of $C^*_{kn_k} > d_i$, a simple interchange of J_{kn_k} and J_{in_i} does not increase the total tardiness, since $p_{kn_k} = \max_j\{p_{jn_j}\}$ and $d_k \leq d_i < C^*_{kn_k}$ by assumption. In both cases, we get another optimal schedule in which either $N_i \rightarrow J_{kn_k}$ or $J_{kn_k} \rightarrow N_i$ holds. Repeatedly applying these two rules to all $i = k+1, \ldots, m$, we must obtain an optimal schedule in which either $J_{kn_k} \rightarrow N_i$ or $N_i \rightarrow J_{kn_k}$ holds for all $i = 1, \ldots, k-1, k+1, \ldots, m$. □

Theorem 7.3 following Theorem 7.2 and Property 7.1 can be constructed as follows.

Theorem 7.3 *Suppose there are m distinct due dates $d'_1 < ... < d'_m$, where $m \ll n$ is a given positive integer, and the jobs in the set N_i of the jobs with due date of d'_i are indexed by the SPT rule as $J_{i1}, ..., J_{in_i}$. Let J_{kn_k} be such a job in N_k that $p_{kn_k} = \max_i\{p_{in_i}\}$. There is some integer δ, $0 \le \delta \le m - k$, such that there exists an optimal schedule π in which J_{kn_k} is preceded by $N_k \setminus \{J_{kn_k}\}$ and all N_i such that $i = 1, ..., k-1, k+1, ..., k+\delta$, and followed by all N_i such that $i > k + \delta$.*

Now, we consider the time requirement of Lawler's dynamic programming algorithm for this special case. Equation (3.1) in Lawler (1977) is recalled as below. We note that the notations in the equation is independent of our notations in Theorem 7.3.

$$\begin{aligned} T(S(i,j,k),t) = \min_{\delta}\{&T(S(i,k+\delta,k'),t) + w_{k'}\max(0, C_{k'}(\delta) - d_{k'}) \\ &+ T(S(k'+\delta+1,j,k'),C_{k'}(\delta))\} \end{aligned} \tag{7.1}$$

where k' is such that

$$p_{k'} = \max\{p_{j'} | j' \in S(i,j,k)\}.$$

First of all, according to Theorem 7.3, each equation (7.1) requires minimization over at most m (not n) alternatives and $O(m)$ running time.

Next, consider any equation (7.1). Taking the definition in Chapter 3, we say the set $S(i,j,k),t)$ is decomposed by job k' and call job k' the decomposition job. Suppose a set $N'_l \subseteq N_l$ are included in S(i,j,k). By Theorem 7.3, all jobs in N'_l or a reduced set $N'_l \setminus \{k'\}$, where k' is the job with the largest index in N'_l and is selected as the decomposition job, should be wholly included in either $S(i, k+\delta, k')$ or $S(k'+\delta+1, j, k')$. As the decomposition job in each recursion is the one with the largest index among the jobs in the selected subset, the set N'_l is always in the form of $\{J_{l1}, J_{l2}, ...\}$. In other words, job i can only be one of the m jobs J_{i1}, $i = 1, ..., m$. Moreover, it is easily seen that, in the case of $i = m$, all jobs in $S(i,j,k),t)$ are from N_m and decomposition for $S(i,j,k),t)$ is unnecessary. Hence, there are no more than $m-1$ values for index i in the set $S(i,j,k)$.

Then, any J_{i1}, $i \leq m-1$, is preceded only by the jobs in N_l, $l < i$. So, the possible values of t are not more than $(n_1+1) \times \ldots \times (n_{m-2}+1) \leq [(n+m-2)/(m-2)]^{m-2}$ in the case of $m > 2$. In the case of $m = 2$, there is only one possible value of t, i.e. $t = 0$.

In conclusion, the problem $1|d_i \in \{d'_1, \ldots, d'_m\}|\sum T_i$ can be solved by Lawler's algorithm in $O(m^2n^2[(n+m-2)/(m-2)]^{m-2}) = O(m^{4-m}n^m)$ time in the case of $2 < m \ll n$.

Finally, we consider the case of $m = 2$. According to Theorem 7.3, in each recursion, if $k' \in N_1$, job k' is either immediately followed by J_{21} or immediately preceded by the job with the largest index among the remaining jobs in N_2, which should also be followed by job k' in the case of $k' \in N_2$. We note that in the case of $k' \in N_2$ and the latter case of $k' \in N_1$, the jobs following job k' are already sequenced in SPT order. So, there are only n_1+1 schedules that need to be considered, i.e. for $j = 1, \ldots, n_1$, the first j jobs in N_1 (in SPT order), followed by the rest jobs in N_1 and all jobs in N_2 in SPT order. As the time requirement of the SPT ordering is in $O(n \log n)$ time, the problem $1|d_i \in \{d'_1, d'_2\}|\sum T_i$ can be solved in $O(n \log n)$ time.

Recall that in the case of m=1, the problem addressed is known as $1|d_i = d|\sum T_i$ and can be solved in $O(n \log n)$ by the SPT rule (Lawler and Moore 1969). We have the following theorem.

Theorem 7.4 $1|d_i \in \{d'_1, \ldots, d'_m\}|\sum T_i$, *$m \ll n$, can be solved in $O(n \log n)$ time in the case of $m = 1, 2$ and $O(m^{4-m}n^m)$ time in the case of $2 < m \ll n$.*

7.4 An extension of the results to the case with release dates

Consider a special case of the problem that we considered in Chapter 5 where the release dates and due dates are strictly agreeable, in the sense that $r_i < r_j$ implies $d_i \leq d_j$ and $r_i = r_j$ implies $d_i = d_j$. By our notation scheme, this problem can be denoted as $1|(r_i, d_i)^=, pmtn|\sum T_i$. By assumption, all jobs simultaneously released are given equal slacks, but the slacks for two jobs with different release dates are unnecessarily equal. It is easy to see that the problem $1|r_i, d_i = r_i + d, pmtn|\sum T_i$ that we consider in Chapter 6 is a special case of $1|(r_i, d_i)^=, pmtn|\sum T_i$. The ordinary *NP*-hardness of this CON due date problem implies that $1|(r_i, d_i)^=, pmtn|\sum T_i$ cannot be polynomially solved. On the other hand, as a special case of $1|(r_i, d_i), pmtn|\sum T_i$, $1|(r_i, d_i)^=, pmtn|\sum T_i$ cannot be strongly *NP*-hard.

Theorem 7.5 $1|(r_i, d_i)^=, pmtn|\sum T_i$ *is* NP*-hard in the ordinary sense.*

Moreover, it is easy to see that the properties identified in the last section hold even in the case that all jobs in a subset N_i, $i = 1, ..., m$ are released simultaneously at r_i, but $r_i < r_{i+1}$ holds for any pair subset N_i and N_{i+1}, $1 \leq i < m$. So, Theorem 7.4 also holds for $1|(r_i, d_i)^=, pmtn|\sum T_i$.

Theorem 7.6 *Let $m \ll n$ be the number of distinct release dates, the problem $1|(r_i, d_i)^=, pmtn|\sum T_i$ can be solved in $O(n \log n)$ time in the case oj $m = 1, 2$ and $O(m^{4-m}n^m)$ time in the case oj $2 < m \ll n$.*

7.5 Conclusions

In this chapter. we have studied the single machine total tardiness problem without release dates. We have proposed an alternative proof for a famous theorem in the literature. We also have investigated a special case where the number of distinct due dates is assumed to be much less than the number of the jobs and extended the results to the case with release dates.

Chapter 8

Conclusions and suggestions

8.1 Conclusions

In this thesis we have presented a review on single machine scheduling with release and due dates and studied several classical single machine scheduling problems with release dates and due dates.

In surveying prior works, the literature has been classified into four classes, namely, maximum lateness, (weighted) number of tardy jobs, total (weighted) tardiness and due date assignment. The review has revealed that, despite the particular importance of the scheduling problems in these classes, a few of them are rarely or never touched and are worthy of research. Such classes include the preemptive scheduling on a single machine to minimize total tardiness with job restrictions, and the due date assignment with release dates.

In Chapter 3, we have presented some definitions for the analysis of the preemptive single machine scheduling problems with a regular criterion, and identified some optimality properties for these problems, especially for the total tardiness problems. Besides, we have proposed a dynamic programming algorithm that is generally applicable.

In Chapter 4, we have studied the problem of preemptive scheduling of equal-length jobs with given release dates on a single machine to minimize total tardiness. We have analyzed some optimal properties for a block with equal-length jobs and presented an $O(n^2)$ algorithm for the overall problem.

In Chapter 5, we have investigated the problem of preemptive scheduling with agreeable release dates and due dates to minimize total tardiness. We proved this problem to be NP-hard and provided a pseudo-polynomial algorithm for it. So, we asserted this problem to be NP-hard in the ordinary sense. Some special cases are identified to be polynomially solvable. Despite our efforts, neither a pseudo-polynomial algorithm nor a reduction from a strongly NP-hard problem is found out for the general version of this problem, in which the release dates and due dates are not always agreeable.

In Chapter 6, we have examined and tackled the class of the due date assignment problems on a single machine with release dates. In each of the problems within this class, n jobs with given release dates should be scheduled on a single machine, while the due dates are assigned to the jobs by the CON/SLK/TWK method. All models with a regular criterion are studied. With a few exceptions, the complexity of most of the problems are explicitly determined.

In Chapter 7, we have studied the single machine total tardiness problem without release dates. We have proposed an alternative proof for a famous theorem in the literature. We also have investigated a special case where the number of distinct due dates is assumed to be much less than the number of the jobs and extended the results to the case with release dates.

8.2 Suggestions

While we have studied some problems in the area of single machine scheduling with release dates and due dates, there are still many issues worthy of consideration for the particular problems that we have studied in this thesis. To extend our research further, we suggest some issues for future work.

1. All problems that we have considered in this thesis are restricted to the single machine environment. It would be of interesting to extend the study to multiple machine environment.

2. In Chapters 4 and 5, we do not consider job weights. From Table 2.3 we know that both the non-preemptive and preemptive versions of the weighted counterpart of the problem considered in Chapter 4 are open to date. It is of great interesting to investigate the complexity of these problems. Though the weighted counterpart of the problem in Chapter 5 is proved strongly $\mathcal{NP}$-hard, the special cases with more complicated job restrictions are still worthy of study.

3. The general version of the problems considered in Chapters 4 and 5, equivalently, the preemptive single machine total tardiness problem with release dates but no other restrictions, is open to the sense of $\mathcal{NP}$-hardness. It is also of interesting to develop a pseudo-polynomial algorithm or a strong $\mathcal{NP}$-hardness proof for this problem.

4. As for the due date assignment problems, all the open problems in Tables 6.2 and 6.3 are worthy of being studied. Besides, the earliness and tardiness models with given release dates are under-explored.

References

Baker, K.R. (1974) *Introduction to Sequencing and Scheduling*, John Wiley & Sons, New York.

Baker, K.R., Lawler, E.L., Lenstra, J.K. and Rinnooy Kan, A.H.G. (1983) Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints. *Operations Research*, **31**, 381-386.

Baptiste, P. (1999a) An $O(n^4)$ Algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Operations Research Letters*, **24**, 175-180.

Baptiste, P. (1999b) Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times. *Journal of Scheduling*, **2**, 245-252.

Baptiste, P. (2000) Scheduling equal-length jobs on identical parallel machines. *Discrete Applied Mathematics*, **103**, 21-32.

Baptiste, P., Brucker, P., Knust, S. and Timkovsky, V. (2002) Fifteen notes on equal-execution-time scheduling, Technical report.

Brucker, P. (2001) *Scheduling algorithms*, Springer, New York.

Brucker, P. and Knust, S. (2002). Complexity results for scheduling problems. *URL: www//mathematic.uni-osnabrueck.de/research/OR/class.*

Chen. B.. Potts. C. and Woeginger. G. (1998) A review of machine scheduling: complexity, algorithms and approximability. *Handbook of Combinatorial Optimization*, Kluwer Academic Publishers, 21-169.

Cheng, T.C.E. and Gordon, V.S. (1994) Optimal assignment of due-dates for preemptive single-machine scheduling. *Mathematical & Computer Modeling*, **20**. 33-40.

Cheng, T.C.E. and Gupta, M.C. (1989) Survey of scheduling research involving due date determination decisions. *European Journal of Operational Research*, **38**, 156-166.

Cheng, T.C.E. and Kovalyov, M.Y. (1999) Complexity of parallel machine scheduling with processing-plus-wait due dates to minimize maximum absolute lateness. *European Journal of Operational Research*, **114**, 403-410.

Chu, C. (1992) A branch-and-bound algorithm to minimize total tardiness with different release dates. *Naval Research Logistics*, **39**, 265-283.

Chu, C. and Portmann, M.C. (1992) Some new efficient methods to solve the $n/1/r_i/\sum T_i$ scheduling problem. *European Journal of Operational Research*, **58**, 404-413.

Du, J.Z. and Leung, J.Y.-T. (1990) Minimizing total tardiness on one machine is *NP*-hard. *Mathematics of Operations Research*, **15**, 483-495.

Emmons, H. (1969) One-machine sequencing to minimize certain functions of job tardiness. *Operations Research*, **17**, 701-715.

Frederickson, G.M. (1983) Scheduling unit-time tasks with integer release times and deadlines. *Information Processing Letters*, **16**, 171-173.

Garey, M.R. and Johnson, D.S. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H.Freeman and Company, New York.

Gordon. V.S. (1993) A note on optimal assignment of slack due-dates in single-machine scheduling, *European Journal of Operational Research.* **70**. 311-315.

Gordon. V.S. and Strusevich. V.A. (1999) Earliness penalties on a single machine subject to precedence constraints: SLK due date assignment. *Computer and Operational Research*, **26**. 157-177.

Gordon, V.S. and Tanaev, V.S. (1983) On minimax problems of scheduling theory for a single machine (in Russian). *Vetsi Akadeii Navuk BSSR. Ser. fizika-matematychnykh navuk*. 3-9.

Gordon. V.S., Roth. J.M. and Chu. C. (2002a) A survey of the state-of-the-art of common due date assignment and scheduling research. *European Journal oj Operational Research*. **139**. 1-25.

Gordon. V.S., Proth. J.M. and Chu, C. (2002b) Due date assignment and scheduling: SLK. TWK and other due date assignment models. *Production Planning & Control*, **13**, 117-132.

Graham, R.L., Lawler, E.L., Lenstra, J.K. and Rinnooy Kan, A.H.G. (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Operations Research*, **5**, 287-326.

Horn, W.A. (1972) Single-machine job sequencing with treelike precedence ordering and linear delay penalties. *SIAM, Journal on Applied Mathematics*, **23**, 189-202.

Jackson, J.R. (1955) Scheduling a production line to minimize maximum tardiness. Research Report 43, Management Science Research Project, University of California, Los Angeles.

Kahlbacher. H.G. and Cheng, T.C.E. (1995) Processing-plus-wait due-dates in single machine scheduling. *Journal of Optimal Theory and Application*, **85**. 163-186.

Kise, H., Ibaraki, T. and Mine., H. (1978) A solvable case of the one-machine scheduling problem with ready and due dates. *Operations Research*, **26**, 121-126.

Kovalyov, M.Y. (1997) Batch scheduling and common due date assignment problem: an *NP*-hard case. *Discrete Applied Mathematics*, **80**, 251-254.

Koulamas, C. (1994) The total tardiness problem: review and extensions. *Operations Research*, **42**, 1025-1041.

Koulamas, C. (1997) Polynomially solvable total tardiness problems: review and extensions. *Omega*, **25**, 235-239.

Koulamas, C. and Kyparisis, G.J. (2001) Single machine scheduling with release times, deadlines and tardiness objectives. *European Journal of Operational Research*, **133**, 447-453.

Labetoulle, J., Lawler, E.L., Lenstra, J.K. and Rinnooy Kan, A.H.G. (1984) Preemptive scheduling of uniform machines subject to release dates. *Progress in Combinatorial Optimization*, Academic Press, New York, 245-261.

Lageweg, B.J. and Lawler, E.L. (1975) Private communication.

Lageweg, B.J., Lenstra, J.K. and Rinnooy Kan, A.H.G. (1976) Minimizing maximum lateness on one machine: computational experience and some applications, *Statistica Neerlandica*, **30**, 25-41.

Lawler, E.L. (1964) On scheduling problem with deferral costs. *Management Science*, **11**, 280-288.

Lawler, E.L. (1973) Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, **19**, 544-546.

Lawler, E.L. (1977) A 'pseudopolynomial' algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, **1**, 31-342.

Lawler, E.L. (1990) 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.

Lawler, E.L. (1994) Knapsack-like scheduling problems, the Moore-Hodgson algorithm and the 'tower of sets' property. *Mathematical and Computer Modeling*, **20**, 91-106.

Lawler, E.L and Labetoulle, A.J. (1978) On preemptive scheduling of unrelated parallel processors by linear programming. *Journal of the Association for Computing Machinery*, **25**, 612-619.

Lawler, E.L. and Moore, J.M. (1969) A functional equation and its application to resource allocation and sequencing problems. *Management Science*, **16**, 77-84.

Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A. H. G. and Shmoys, D. B. (1993) Sequencing and scheduling: algorithms and complexity. *Logistics of production and inventory*, North-Holland, Amsterdam, Holland.

Lee, C.-Y., Uzsoy, R. and Martin-Vega, L.A. (1992) Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, **40**, 764-775.

Lenstra, J.K. -unpublished.

Lenstra, J.K. and Rinnooy Kan, A.H.G. (1980) Complexity results for scheduling chains on a single machine. *European Journal of Operational Research*, **4**, 270-275.

Lenstra, J.K., Rinnooy Kan, A.H.G. and Brucker, P. (1977) Complexity of machine schedule problems. *Annals of Discrete Mathematics*, **1**, 343-362.

Leung, J. Y.-T. and Young, G.H. (1990) Minimizing total tardiness on a single machine with precedence constraints. *ORSA Journal on Computing*, **2**, 346-352.

Papadimitriou, C.H. (1998) *Combinatorial optimization: algorithms and complexity*, Dover, New York.

Pinedo, M. (1995) *Schedule: Theory, Algorithms, and Systems*, Prentice Hall, Jersey.

Qi, X., Yu, G. and Bard, J.F. (2001) Single machine scheduling with assignable due dates. *Discrete Applied Mathematics*, 122, 211-233.

Sahni, S. (1979) Preemptive scheduling with due dates. *Operations Research*, **27**, 925-934.

Schrage, L. (1968) A proof of the shortest remaining processing time processing discipline. *Operations Research*, **16**. 687-690.

Simons, B. (1978) A fast algorithm for single processor scheduling. *Proceedings of the 19th IEEE symposium on Foundations of Computer Science*, 246-252.

Smith, W.E. (1956) Various optimizers for single-stage production. *Naval Research Logistic Quarterly*, **3**, 59-66.

Szwarc, W., Croce, F.D. and Grosso, A. (1999) Solution of the single machine total tardiness problem. *Journal of Scheduling*, **2**, 55-71.

Szwarc, W., Grosso, A. and Croce, F.D. (2001) Algorithmic paradoxes of the single-machine total tardiness problem. *Journal of Scheduling*, **4**, 93-104.

Tanaev, V.S. and Gordon, V.S. (1983) On scheduling to minimize the weighted number of late jobs (in Russian). *Vestisi Akad. Navuk Belarus Ser. Fizi.-Mat. Navuk*, **6**, 3-9.

Tanaev, V.S., Gordon. V.S. and Shafransky. Y.M. (1994) *Scheduling theory: Single-stage systems.* Kluwer. Boston.

Appendix

Publications

Refereed Journal Articles:

Tian, Z.J., Ng, C.T. and Cheng, T.C.E. (2001) Preemptive scheduling with agreeable due dates to minimize total tardiness. *Proceedings of the* 5th *International Conference on Optimization: Techniques and Applications*, 405-412.

Tian, Z.J., Ng, C.T. and Cheng, T.C.E. (2001) An $O(n^2)$ algorithm for scheduling equal-length preemptive jobs on a single machine to minimize total tardiness. Accepted by *Journal of Scheduling.*

Tian, Z.J., Ng, C.T. and Cheng, T.C.E. (2002) Preemptive scheduling with agreeable release dates and due dates to minimize total tardiness. Submitted to *Annals of Operations Research.*

Tian, Z.J., Ng, C.T. and Cheng, T.C.E. (2002) Single machine scheduling with release dates and CON/SLK/TWK due dates. Submitted to *Computers and Operations Research.*

Tian, Z.J., Ng, C.T. and Cheng, T.C.E. (2002) On the single machine total tardiness problem. Submitted to *European Journal of Operational Research.*

Conference Presentation:

Tian, Z.J., Ng, C.T. and Cheng, T.C.E. (2001) Preemptive scheduling with agreeable due dates to minimize total tardiness. *The 5th International Conference on Optimization: Techniques and Applications*, Hong Kong.