An FPTAS for Total Weighted Earliness Tardiness Problem with Constant Number of Distinct Due Dates and Polynomially Related Weights

AN FPTAS FOR TOTAL WEIGHTED EARLINESS TARDINESS PROBLEM WITH CONSTANT NUMBER OF DISTINCT DUE DATES AND POLYNOMIALLY RELATED WEIGHTS

By

Jingjing Huang, B.Eng.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Science

McMaster University

MASTER OF SCIENCE (2013) McMaster University
(Computing and Software) Hamilton, Ontario

TITLE: An FPTAS for Total Weighted Earliness Tardiness Problem with Constant Number of Distinct Due Dates and Polynomially Related Weights

AUTHOR: Jingjing Huang, B.Eng. (Uni. of Science and Technology of China)

SUPERVISOR: Dr. George Karakostas

NUMBER OF PAGES: x, 72.

Abstract

We are given a sequence of jobs on a single machine, and each job has a weight, processing time and a due date. A job is early when it finishes before or on its due date and its earliness is the amount of time between its completion time and its due date. A job is tardy when it finishes after its due date and its tardiness is the amount of time between its due date and its completion time. The TWET problem is to find a schedule which minimizes the total weighted earliness and tardiness. We are focusing on the TWET problem with a constant number of distinct due dates and polynomially related weights. This problem has been proven to be NP-hard. In this thesis, we present a dynamic programming algorithm for our TWET problem first and then convert it into an FPTAS by adopting a rounding scheme.

There are several important points in our algorithm: we observe the importance of the straddlers and guess them at the beginning through exhaustive enumeration, and insert them back at the very end by solving a linear problem; we know a series of structural properties of the optimal schedule to shrink the state space of the DP; we increase each due date to get a new problem and adopt a rounding scheme of the DP for the new problem to avoid preemption. Finally we move the due dates back to get the final schedule for the original TWET problem without changing the objective value much.

Acknowledgments

First of all, I gratefully acknowledge my supervisor, Dr. George Karakostas. His advice and supervision during the regular meetings between us helped me so much with my research. His serious attitude towards the research and broad horizon of knowledge impressed me deeply to make myself more concentrated and enthusiastic on my work. Besides, his patience for pointing my grammar errors when we were talking helped me improve my oral English a lot.

I wish to express my sincere thanks to Dr. Antoine Deza, Dr. Franya Frannek and Dr. Sanzheng Qiao. Their courses gave me a deeper understanding of algorithms and computation theory which helped and inspired me with my research in scheduling problems. My special thanks are to the members of the examination committee: Dr. Alan Wassyng, Dr. Franya Franek and Dr. George Karakostas.

Many thanks to my friend Kun Hu, our discussions and his suggestions helped me with my work. Also thanks to my friends Linyan Liu, Haibo Liang, Chao Zhu, Yunfei Cai, Yi Hu, Linna Pang, Xiang Yin, Bingzhou Zheng and so on. They are helpful and supportive and I had a great time with them.

At last, I would like to thank my parents in China. With their supports and encouragements, I completed my thesis program successfully.

Contents

Abstract **iii**

Acknowledgments **v**

List of Figures **ix**

1 Introduction **1**

1.1 The TWET Problem and Motivation 1

1.2 Previous Work 4

1.2.1 Previous work on TWT Problem 4

1.2.2 Previous work on TWET Problem 7

1.3 Our Work 8

1.4 Thesis Outline 12

2 Approximation Algorithms and Some Scheduling Problems **15**

2.1 Overview of Approximation Algorithms 15

2.1.1 Basic Concepts 15

2.2 Structural Properties in Single Machine Scheduling 18

2.3 Preemption Properties for the TWET Problem 21

3 FPTAS for the Single Machine TWET Problem with Common Due Date **25**

3.1 FPTAS for TWT Problem with Common Due Date 25

3.1.1 Preliminaries 25
3.1.2 DP for the Non-straddling Jobs 26
3.1.3 Inserting the Straddler Back 27
3.1.4 Rounding Scheme 28
3.2 FPTAS for TWET with Common Due Date 29
3.2.1 Preliminaries 29
3.2.2 DP for the Non-straddling Jobs 31
3.2.3 Symmetric Quadratic Knapsack Problem 32
3.2.4 Rounding Scheme 34
3.2.5 Scheduling Without a Straddler 35
3.2.6 Scheduling With a Straddler 35

4 FPTAS for the Single Machine TWET Problem with Distinct Due Dates and Polynomially Related Weights 39
4.1 Preliminaries 39
4.2 An Exact Dynamic Programming Algorithm 42
4.3 The FPTAS 45
4.3.1 The Algorithm with Rounding Scheme 45
4.3.2 Proof of Near Optimality 47
4.3.3 Complexity of the Rounding Algorithm 54
4.4 A Simpler Rounding Scheme 55
4.4.1 A New Dynamic Programming 55
4.4.2 The New FPTAS 57
4.4.3 General Straddlers 62

5 Conclusions and Open Questions 65
5.1 Conclusion 65
5.2 Open Questions 66

Bibliography 67

List of Figures

4.1 Insert job J_k as a tardy job in interval I_i 41
4.2 Insert job J_k as an early job in interval I_i 41
4.3 The original due dates and intervals 46
4.4 The new due dates and intervals 46
4.5 Algorithm FPTAS 47
4.6 The Simpler FPTAS Algorithm 58

Chapter 1

Introduction

Scheduling, with the objective to minimize (or maximize) one or more performance measures, is the allocation of limited resources to various activities [30]. Scheduling problems involve both single machine and multi-machine scheduling problems. If the single machine problem can be solved, its solution can be used to understand and provide the basis to solve the more complex multi-machine problem. The Just-In-Time(JIT) scheduling is an important branch of scheduling in which each scheduling task needs to be executed as close to its due date as possible. In this thesis, we are particularly interested in the problem of minimum total weighted earliness-tardiness (TWET) in JIT scheduling on a single machine.

1.1 The TWET Problem and Motivation

The goal of scheduling tasks in a Just-In-Time (JIT) fashion has been a central goal for logistics. According to JIT, the minimum total weighted earliness-tardiness problem (TWET) for a single machine is defined as follows. Based on a single machine environment, we are given n jobs indexed as $J_j, j = 1, 2, \ldots, n$. Assume that all the n jobs are ready at time 0 for processing and no preemption is allowed. Each job has a processing time p_j and due date d_j by which the processing of job J_j is due to be completed. We want to schedule all the n

jobs as close to their due dates as possible. Once a job cannot be executed to complete exactly at its due date, there is a penalty measured as the weight w_j (per unit of time) of the job J_j and it shows how important the job is.

With the fact that only one job can be processed at a time on a single machine, the processing order together with the time that each job finishes at of the n jobs identifies one specific scheduling. For a given scheduling (processing order) S, the completion time of job $J_j, j = 1, 2, \ldots, n$ is defined by $C_j(S)$ or just C_j. Therefore, two quantities related to the completion time $C_j(S)$ of job J_j in a schedule S are important: its *tardiness* $T_j(S) := \max\{0, C_j(S) - d_j\}$, and its *earliness* $E_j(S) := \max\{0, d_j - C_j(S)\}$. A job is called a *tardy* job when its tardiness is positive. Otherwise, it's an *early* job. We say that each job j weights both its earliness and tardiness *symmetrically*, i.e., with the same weight w_j. We study the symmetric weights in the thesis. Then motivated by the logistical objective, the minimum *total weighted earliness tardiness* (TWET) problem on a single machine is defined as the computation of a schedule S on this machine that minimizes the total weighted earliness and tardiness $\sum_{j=1}^{n} w_j(E_j(S) + T_j(S))$, or simply $\sum_{j=1}^{n} w_j(E_j + T_j)$.

For all $j = 1, ..., n$, the due date d_j value comes from a set of K possible distinct due dates $\{d_1, d_2, ..., d_K\}$, while $d_1 < d_2 < \cdots < d_K$, $1 \leq K \leq n$. Let $W := \sum_j w_j$, and $w_{min}(w_{max})$ be the minimum(maximum) job weight. In this thesis, we are focusing on the above TWET problem with a constant number of distinct due dates and polynomially related symmetric weights, i.e., based on the following

Assumption 1.1.1 *(i) K is a* constant*, (ii) the weights of the jobs are* polynomially related*, i.e.,* $\frac{w_{max}}{w_{min}} = O(poly(n))$.

A little different from the definition of TWET problem above, the total weighted tardiness (TWT) problem only considers the tardy jobs, i.e, all the early jobs have no cost and only tardy jobs count for the total cost. Thus, the minimum *total weighted tardiness* (TWT) problem on a single machine is defined as the computation of a schedule S on this machine that minimizes the total weighted tardiness $\sum_{j=1}^{n} w_j T_j$. Many results have been proposed for the TWT problem in, e.g., [22], [21], [20]. It is obvious that the TWT problem is a special case of TWET problem when the weight for early jobs is 0. Thus, we also introduce the TWT problem to provide some basic idea for our problem.

The TWET problem has attracted the attention of both industry and academic researchers. In a JIT scheduling environment, jobs that complete early must be held in goods inventory until their due dates, while jobs that complete tardy may cause a customer to shut down operations. Thus, many scheduling problems, like goods deliveries, products on assembly lines in factories, can be modeled as a TWET problem of JIT scheduling. It has received much attention not only because it is meaningful in practice, but also because it has inspired several new methods in the design of the solution procedures through a non-regular performance measure. All these problems are known to be NP-hard. If we can solve the TWET problem, then a series of related problems can be solved too.

Because of the hardness of the TWET problem with arbitrary due dates and arbitrary weights, there're only a few results. So far, only some special cases of the TWET problem have been intensively researched, for instance, when all the jobs have the same processing time [38], [39]; when no idle time is allowed between jobs [1]; when all the jobs have a common due date [23]; when all the weights are the same, and so on. In this work, we tackle the TWET problem for the case of a constant number K of distinct due dates and polynomially related weights.

1.2 Previous Work

In this section, we will introduce several previous works in both TWT and TWET problem by which our work is inspired. Although both of them have been proven to be strongly NP-hard for arbitrary weights [27][29][7], various aspects of solving the TWT and TWET problem and their variants have attracted considerable attention from many researchers. We present some of the results first and we only cite a limited number of them since they are numerous (see, e.g. [2], [7], [14]).

1.2.1 Previous work on TWT Problem

Several methods have been applied to solve the minimum TWT problems. We mainly introduce the typical branch-and-bound method, dynamic programming method and approximation algorithms in turn.

The branch-and-bound method was firstly used to TWT problem in [11]. In this paper, Elmaghraby showed the branch-and-bound method is more efficient than the dynamic programming method. Then Shwimer [36] proposed a branch-and-bound algorithm for the TWT problem which is applicable on a computer. Since then, the branch-and-bound technique for TWT problem has been widely and deeply researched.

Picard et al. [31] used the method of branch-and-bound and sub-gradient optimization to find the shortest paths in a network when they observed that the time-dependent traveling salesman problem can be modeled as the TWT problem.

To restrict the size of the search and find the optimal solution, Emmons [12] developed dominance rules. Based on the dominance rules, Rachamadugu [34] showed a local dominance property among adjacent jobs in an optimal schedule. Importantly, such a proposition identified the weighted shortest processing time (WSPT) order among adjacent tardy jobs in the optimal schedule of TWT problem. The WSPT order is important through the whole thesis.

Potts et al. [33] formulated the TWT problem as a Lagrangian problem and used the local dominance property in [34] in their branch-and-bound algorithm. They used the Lagrangian dual solution as a lower bound for the TWT problem. However, such a lower bound to find the optimal solution was not practical to use because of the extensive computation the algorithm needs. Thus, Akturk et al. [3] introduced a new dominance rule to develop a new lower bound. The new dominance rule was a generalization of the rules in [12] and [29], and it could reduce the number of alternatives for finding the optimum to make the algorithm more efficient.

More recently, Babu et al. [5] developed a Lagrangian decomposition on a $0-1$ time indexed formulation and then used the optimal value of the duality problem as a lower bound. They also designed a Lagrangian heuristic to get an upper bound. They gave the branch-and-bound algorithm using the lower and upper bound. They also combined the dominance and elimination rules to get a trade-off between a tighter lower bound and the time needed for the enumeration. Up to now we have introduced several works using the branch-and-bound method and the shortcoming of it is that it needs a huge amount of computation time.

For the dynamic programming method, a quite early use in scheduling problems was proposed by Schrage et al. [35]. They considered several constraints from practical considerations and from the characteristics of the optimal schedule, to reduce the number of possible sequences. Based on the enumeration of all feasible subsets of tasks, they presented a method to assign an easily computed label to each feasible subset. As a consequence, a dynamic programming algorithm can be applied to many scheduling problems, including the TWT problem.

Arkin et al. [4] proposed a pseudo-polynomial time algorithm using dynamic programming to solve a special case of TWT problem where the weight of each job is proportional to its processing time. Huegler and Vasko [19] developed a dynamic programming based on heuristics. A big improvement has been made to reduce the time complexity by Congram [10]. He proposed a new neighborhood search technique which is called dynasearch. Dynasearch uses dynamic programming in local search. It can explore exponential size neighborhoods in polynomial time and hence it decreases the computation time. Although the dynamic programming method can generate the exact solution, it is still constrained by the computer storage and its time complexity.

Not only branch-and-bound and dynamic programming method have been extensively researched, but also approximation algorithms have been developed to solve scheduling problems. With certain degree of loss in optimality, approximation algorithms can compute the near optimal solution quite efficiently. Nowadays, approximation algorithms are receiving increasing attention for solving combinatorial optimization problems. For the NP-hard problem, a fully polynomial time approximation scheme (FPTAS)[1] is the strongest possible result. We give the details of FPTAS in section 2.1.

Lawer [27] presented an FPTAS for the TWT problem when the weights of all the jobs are the same through a modification of the dynamic programming algorithm. When the differences between the due dates of jobs and their processing times are the same constant, i.e., $d_i = p_i + q$ where q is a constant, Cheng et al. [9] constructed an FPTAS. Besides, when there's a common due date for all the jobs, Kellerer and Strusevich [22] give an FPTAS by applying a rounding scheme to the dynamic programming method.

When the number of distinct due dates is a constant, Kolliopoulos et al. [25] presented a pseudo-polynomial time dynamic programming algorithm. With the assumption that the maximal job weight is bounded by a polyno-

[1]An FPTAS is a family of algorithms that given any constant $\varepsilon > 0$ produce a solution within a factor of $(1 + \epsilon)$ of the optimal in time polynomial in n and $1/\epsilon$.

mial in n, where n is the total number of input jobs, they used the rounding scheme in [28] for their dynamic programming algorithm to generate an FPTAS. Karakostas et al. [21] also proposed an FPTAS for the more general case with a fixed number of distinct due date.

1.2.2 Previous work on TWET Problem

Due to the significance of TWET problem, it has a long history of intensive study (see, e.g., [15] and [18]). Its NP-completeness was proven (quite non-trivially) by Garey et al. [13], and recently Müller-Hannemann and Sonnikow [37] showed that the weighted version is notoriously hard, unless $P = NP$. It cannot be approximated within a factor $O(b^n)$ for any constant $b > 0$ in polynomial time. Nevertheless, there have been good exact or approximation algorithms for several important special cases.

Garey et al. [13] show that the unweighted version with all jobs having the same processing time can be solved in $O(n \log n)$ time.

For arbitrary weights and arbitrary number of due dates, quite little is known on the approximation of TWET. When all the jobs have the same processing time, Verma and Dessouky [39] give a linear programming exact solution in polynomial time. Müller-Hannemann and Sonnikow [37] give a constant ratio approximation algorithm for *constantly* related weights and processing times, i.e., the ratio between the maximal processing time p_{max} and the minimal one p_{min} among all jobs is a constant, and, likewise, the ratio between the maximal weight w_{max} and the minimal one w_{min} is also a constant; for a constant number of due dates, through guessing all the possible straddles (defined as the jobs which start before a due date and finish after a due date in their paper) and the starting time of each straddler and then inserting all the other jobs using dynamic programming, they give a pseudo-polynomial algorithm whose complexity depends on the total processing time to solve the problem exactly.

When all the jobs have a common due date, there are two possibilities: the *non-restrictive* case, where $d \geq \sum_j p_j$, and the harder *restrictive* case, when $d \leq \sum_j p_j$. The former is still NP-complete for the weighted case [16], but there is a fully-polynomial time approximation scheme (FPTAS) for this case [26]. This scheme does not require any prior knowledge of lower and upper bounds on the value of a complete optimal solution since it recursively computes lower and upper bounds on the value of partial optimal solutions. More recently, Kellerer and Strusevich [23] presented an FPTAS for the more difficult restrictive case via a connection to the quadratic knapsack problem. We will introduce their work in Chapter 3.

1.3 Our Work

In this thesis, we present an FPTAS for the TWET problem with a constant number K of distinct due dates and polynomially related weights given in Section 1.1 and it's proven to be NP-hard [7]. We design a pseudo-polynomial algorithm using dynamic programming first and then apply a rounding scheme to obtain the desired approximation scheme. This work is inspired by [23], [21], but we introduce a number of new ideas for our problem.

To understand our algorithm, an important thing is to understand the intrinsic difficulty of the TWET problem which lies in the change of objective value when the completion time of a job slides from before its due date (being early) to after it (being tardy). We note that the jobs in an optimal schedule that straddle the due dates (or exactly finished at some due date), defined as *straddlers*, play a special role in splitting the problem into a number of 'knapsack' problems (one per interval between two consecutive due dates, except the last (infinite) interval). This indicates that we should distinguish the straddlers from all the other jobs. A common assumption is that every straddler is only straddling one due date and we can see that each due date has a straddler (straddling the due date or finishing exactly on the due date). In reality, we have no idea about the optimal schedule. Thus, knowing which job is the straddler for each due date becomes a problem. A useful and accurate technique, although not that efficient, is exhaustive enumeration (or guessing). By guessing, we mean that, we enumerate all possible combinations of the straddlers from the set of all jobs. We show that this guessing can be done in polynomial time when the number of distinct due dates is constant in the following chapters. And we can say we "have guessed" the straddlers for each due date from now on.

Once the straddlers for all due dates have been guessed, they divide the time horizon into intervals. We notice that in each interval, all the tardy jobs must be processed before all the early jobs since both of them contribute to the objective value. To minimize the total earliness and tardiness, it has been proved in [7] that these early jobs must be in I-WSPT (inverse weighted-shortest-processing-time, i.e., $\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \cdots \geq \frac{p_n}{w_n}$) order and the tardy jobs must be in WSPT order. This fact compels us to sort the input jobs at the beginning in I-WSPT order in our algorithm. We sort the jobs before inserting them to avoid keeping record of which job goes where. Instead, we only need to record the total earliness and tardiness in each interval.

The next step is to consider the placement of the jobs including the tardy straddlers. We follow the idea of Kellerer's and Strusevish's rounding scheme to produce an FPTAS [23] in which they insert the straddlers after inserting all the other jobs. Then how to place the other jobs and where to place the straddler become the problems. Here, we can think backwards: Suppose we have the optimal schedule and we can know which jobs are straddlers for the due dates. If we remove those straddlers, we are left with a hole around each due date to separate the early and tardy jobs. Now we can push the early jobs forward and tardy jobs backward to the due date to remove the hole between them (actually, moving the place of the hole) for each due date. We notice that all the early jobs are still early and the tardy jobs are still tardy, although each of them has a decrement in its earliness (or tardiness) by the same amount. Now we are left with a partial schedule without the straddlers. Note that the placement of these jobs are exactly the same as their placement in the optimal schedule. The difference is only the earliness or tardiness of each job. With the partial schedule, the optimal schedule can be constructed by inserting each straddler between the early and tardy jobs for each due date optimally. This optimal insertion of the straddlers can be expressed as a linear problem (LP) which can be solved in polynomial time.

Practically, since we do not know which partial schedule will give us the optimal schedule, we have to try all possible placements in order to find it. When inserting a job in a interval, we must check whether there is enough space for it. We come up with sufficient feasibility conditions to check that. When a job is to be inserted as early (or tardy) in a interval, there must be enough space in the interval such that this insertion will not push any previously inserted jobs into any other intervals. Besides, since the straddles need to be inserted in the end, we still need a condition to make sure that we can insert the straddlers for all the due dates without making any inserted early job tardy or tardy job early. Once all these conditions are satisfied, a job can be inserted feasibly and safely. More specifically, before inserting a non-straddling job, we check if there is enough space for it first, and then we check if the straddlers can still be inserted after the insertion of this job.

Since we are going to use a rounding scheme in the FPTAS, the feasibility conditions must also hold for the FPTAS. Here we play a trick: we increase each due date (except the first one) a little to make the space of the intervals between every two adjacent new due dates bigger than that between the original ones.

This is the big difference from [21] since no preemption is allowed in our TWET problem and it is a new idea. To reduce the running time of the algorithm, we round up the summation of total processing time of early and tardy jobs from the second interval to the last interval. One consequence of this rounding up is that some jobs which are early (or tardy) in the optimal schedule cannot be inserted as early (or tardy) in the rounded schedule. The small extension of the intervals can prevent that without any preemption. We then use the new due dates to insert the jobs and the straddlers. At the very end, we move the due dates back (without moving the jobs) to their original values with a possible increase in objective value to get the final schedule. We can prove that such movements only contribute a little to the objective value since the extensions are small.

In order to make the complexity of the algorithm polynomial in the problem size, we have to consider a reduced number of values of the variables by rounding up. Of course, we can not reduce them arbitrarily. We have to confine our output within a neighborhood of the optimal solution. It is the feasibility conditions that help us to achieve this goal. Applying the FPTAS to the TWET problem with common due date [23], or even the TWT problem with common due date [22] and distinct due dates [21], it's easy to see that our algorithms can solve them with small changes. We develop an FPTAS for the TWET problem with a constant number of distinct due dates and polynomially related weights. When the number of distinct due dates is arbitrary or the weights are arbitrary, the complexity of the algorithm will be exponential.

To summarize, our FPTAS algorithm works as follows: Guess K (K is number of distinct due dates) straddlers (exhaustive enumeration) when the straddler is defined as the job starting before a due date but finishing after or on the same due date. Order the remaining $m = n - K$ jobs in inverse weighted shortest processing time (I-WSPT) order, i.e., $\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \cdots \geq \frac{p_m}{w_m}$.

Then move the due dates a little bigger to get the new due dates. For each set of the guessed straddlers with the new due dates, we apply the following dynamic programming for the non-straddling $m = n - K$ jobs:

- Each job can be inserted either early or tardy by selecting the proper interval and it needs to be inserted without any preemption. Before inserting the job, we need to check the feasibility conditions to make sure that there is enough space in the interval to insert that job. After the insertion, the feasibility conditions must also hold for the straddlers. The job can be inserted successfully only when all the conditions are satisfied. If a placement can be done, we round the states of this DP by (i) grouping them into groups with very similar characteristics (see Algorithm FPTAS given in Chapter 4), and (ii) keeping the state that maximizes the total processing time packed in the (finite) intervals except the first. In this way, there is a sequence in our rounded DP that follows closely the optimal DP (given as a Lemma in Chapter 4). The heart of our algorithm is the idea that we can do (i) and (ii) *not* for the original problem, but for the problem where we have given our schedule a little more space by moving the due dates later (but not much later).

After inserting all the non-straddling jobs, we insert the straddlers into their optimal positions by solving a linear problem. And finally, we show that moving the new due dates back to their original values doesn't increase the objective value by much. This FPTAS algorithm is the main contribution of this thesis.

1.4 Thesis Outline

In this chapter, we have given a brief introduction of the TWET problem, the related backgrounds and the previous works. The remainder of this thesis is organized as follows. Basic background of approximation algorithms and some related scheduling problems will be introduced in Chapter 2. In Chapter 3, we introduce the key ideas of [23] which are important for our work. We present the main result of this thesis in Chapter 4, an FPTAS for the TWET problem with a constant number of distinct due dates and polynomially related weights. Then in the last Chapter, we give a summary of the whole thesis and possible future work.

Chapter 2

Approximation Algorithms and Some Scheduling Problems

From the introduction in Chapter 1, we know that approximation algorithms can be used to solve NP-hard scheduling problems. Here we say that we "solve" the problem if an algorithm can give a solution within the approximation guarantee. In this chapter we will introduce some basic concepts of approximation algorithms. Besides, we notice that some basic scheduling problems are related to each other. If we find an optimal solution for one problem, it may help us to find an optimal schedule for another, just like the solution for TWET with one common due date can help us get a solution for TWET with distinct due dates. Thus, we also introduce several elementary results and properties of optimality for some scheduling problems.

2.1 Overview of Approximation Algorithms

2.1.1 Basic Concepts

Many natural optimization problems, which contain the scheduling problems we introduce in Chapter 1, have been proven to be NP-hard. Therefore, under the wide common belief that $P \neq NP$, the exact optimal solutions are usually prohibitively time consuming. If the exact optimal value is not required, we will be satisfied with a provably good solution produced by an efficient algorithm. That is the idea of approximation algorithms, which is, generate provably near-optimal solutions quite efficiently in terms of complexity of algorithms. Approximation algorithms have attracted increasing attention and become more prominent as the tool for the generation of the near optimal solutions for NP-hard optimization problems since the mid-20th century.

In the following paragraphs, we will list several exact mathematical definitions of the main concepts in the area of approximation algorithms. Details can be found in [40]. An optimization problem is specified by an input set $\mathcal{I}$, a set $S(I)$ of feasible solutions (i.e. the solutions which satisfy all constraints) for each input $I \in \mathcal{I}$, and an objective function f that calculates an objective value $f(s)$ for every feasible solution $s \in S(I)$. We assume that all feasible solutions have non-negative objective values. We denote the optimal objective value for input I by $OPT(I)$ and denote the size of an input instance I by $|I|$, i.e., the number of bits used in writing down I in some fixed encoding. Since the TWET problem we are focusing on is NP-hard, it is impossible to find the exact optimal solution within polynomial time in $|I|$ (unless $P = NP$) except in special cases.

$\rho-$Approximation Algorithm

Let **P** be a minimization problem. Let $\rho > 0$. An algorithm A is called a $\rho-$approximation algorithm for problem **P**, if for every instance I, it returns a feasible solution with objective value $\hat{f}(s)$ such that

$$\hat{f}(s) - OPT(I) \leq \rho \cdot OPT(I)$$

The value ρ is called the performance guarantee or the worst case ration of the approximation algorithm.

Pseudo-Polynomial Time Approximation Algorithms

As described above, an algorithm is said to run in polynomial time if its running time is polynomial in the size of the input instance $|I|$, the number of bits needed to write I.

Definition 2.1.1 *Pseudo-Polynomial Time is polynomial time when the numeric values in the input string (given as integers) are given in unary representation rather than in binary.*

A Pseudo-Polynomial Time Algorithm is an algorithm if its running time is polynomial in the size of input instance $|I|$ and the numeric values of the input (provided these are given as integers), rather than rather than the binary representation of their values.

Polynomial Time Approximation Schemes

Definition 2.1.2 *A Polynomial Time Approximation Scheme (PTAS) for a problem* $\boldsymbol{P}$ *is a collection of* $1+\epsilon$ *approximation algorithms* $A(\epsilon)$*, one for every constant* $\epsilon > 0$*, whose time complexity is polynomial in the input size* $|I|$.

Fully Polynomial Time Approximation Schemes

Definition 2.1.3 *A Fully Polynomial Time Approximation Scheme (FPTAS) for a problem* $\boldsymbol{P}$ *is a PTAS whose time complexity is polynomial in both the input size* $|I|$ *and* $\frac{1}{\epsilon}$.

From the above definitions, the only difference between FPTAS and PTAS is the demand on polynomial time in $\frac{1}{\epsilon}$. In terms of the worst case approximation, an FPTAS gives the strongest possible result that we can hope to derive for an NP-hard problem.

From the definition of the approximation algorithm, the near-optimal output needs to be compared with the optimal objective value to make sure that the ratio between them is within the approximation ratio. However in reality, for a specific minimization problem, we have no idea about what its optimal objective value is, or how to find the optimum in polynomial time. Thus, we try to find a lower and an upper bound for the optimal solution of the problem instead of finding the optimum itself and relate the approximate near-optimal output to these bounds. We will show how to chose such bounds for the approximation scheme for our TWET problem in Chapter 3 and Chapter 4.

In our TWET problem, the input is the number of the jobs n, each job's weight w_j and processing time p_j, $j = 1, 2, \ldots, n$ and the K distinct due dates $d_1 < d_2 < \cdots < d_K$. If the enumeration method is used, we need to evaluate all the possible $n!$ sequences to find the optimal schedule. We can also use dynamic programming in this case. It is typically more efficient than brute-force enumeration even though the complexity of dynamic programming grows at an exponential rate with the increase of the problem size. The difference between the two methods is that the dynamic programming considers certain ordering sequences indirectly while enumeration considers all possible sequences explicitly. We can get the optimal using dynamic programming, but it still takes exponential time. Thus, we need to come up with an approximation algorithm which uses dynamic programming, but constraints the sequences (i.e. number of states) to be polynomial.

2.2 Structural Properties in Single Machine Scheduling

There are several special structural properties in the optimal schedule for some scheduling problems. For example in [8], the Shortest Processing Time (SPT) rule is used for the Minimum Mean Flow-time problem; the optimal sequence for the Maximum Lateness problem follows the Earliest Due Date (EDD) rule;

the optimal sequence for the Minimum Total Weighted Completion Time problem and the TWT problem is in the Weighted Shortest Processing Time (WSP-T) order. In these cases, the optimal schedule can be constructed through a simple sorting based on these special ordering rules. These results can also be used directly to find the solution of some practical scheduling problems in certain situations. Besides, we can also use them to understand the the optimal schedule for other more complicated scheduling problems, such as the TWET problem we are dealing with.

In this section, we introduce the WSPT rule since an optimal schedule for the TWET problem is constructed from a combination of WSPT and I-WSPT which is the inverse order of WSPT. For purpose of completion, we include the proofs together with the lemmas. For more details, one is referred to [6]. The minimum total weighted completion time is defined as

$$\min_{schedule\ of\ n\ jobs} \{\sum_{j=1}^{n} w_j C_j\} \tag{2.2.1}$$

where C_j is the completion time of job J_j.

Weighted Shortest Processing Time Order

Definition 2.2.1 *Given n jobs, each job J_j has a processing time p_j and a weight w_j, $1 \leq j \leq n$. The Weighted Shortest Processing Time (WSPT) order is to sort the jobs according to the processing time per weight such that the first job has the shortest processing time per unit of weight, the second has the second shortest processing time per unit of weight, and so on, that is,* $\frac{p_1}{w_1} \leq \frac{p_2}{w_2} \leq \cdots \leq \frac{p_n}{w_n}$.

We have the following lemma:

Lemma 2.2.1 *The optimal schedule for the single the single machine minimum total weighted completion time problem, as defined in equation (2.2.1) follows the WSPT order.*

Proof. (from [6]) By contradiction.
Suppose an optimal schedule $\mathcal{S}$ is not in WSPT order. In this schedule, there must be at least two adjacent jobs, J_i and J_j, with J_j following J_i, such that $\frac{p_i}{w_i} > \frac{p_j}{w_j}$. Assume job J_i starts at time t.
Now construct a new schedule $\mathcal{S}'$ after performing an adjacent pairwise interchange on job J_i and J_j. In the new schedule $\mathcal{S}'$, job J_j starts at time t and is followed by job J_i, while in the original schedule $\mathcal{S}$, job J_i starts at time t and is followed by job J_j. Every other job remains in its original position. It's easy to see that the total weighted completion time of the jobs processed before jobs J_i and J_j is not affected by the interchange. Neither is the total weighted completion time of the jobs processed after jobs J_i and J_j. Thus the difference in the objective values of the two schedules $\mathcal{S}$ and $\mathcal{S}'$ lies only in jobs J_i and J_j. Under $\mathcal{S}$, the total weighted completion time of jobs J_i and J_j is $(t + p_i)w_i + (t + p_i + p_j)w_j$, while under $\mathcal{S}'$, it is $(t+p_j)w_j + (t+p_i+p_j)w_i$. It's easy to verify that $(t+p_j)w_j + (t+p_i+p_j)w_i$ is strictly less than $(t+p_i)w_i + (t+p_i+p_j)w_j$ when $\frac{p_i}{w_i} > \frac{p_j}{w_j}$. This contradicts the optimality and the proof is done. □

With the lemma above, we can easily get the following corollary which is a very important structural property in minimum TWET problem with common due date.

Corollary 2.2.1 *In any optimal schedule of the minimum TWET problem with common due date, the tardy jobs must appear in WSPT order while the early jobs appear in inverse WSPT (I-WSPT) order.*

2.3 Preemption Properties for the TWET Problem

For the TWET problem given in Section 1.1, no preemption is allowed, i.e., if one job starts to be processed, we must wait until it finishes to process the next job. If preemption is allowed, i.e, a job can be divided into pieces and the pieces can be processed non-contiguously, it gives us a broader horizon to investigate the structural properties of the optimum. This may help us in future work. We say a preemption scheme is 'allowed' when the TWET problem with such a preemption outputs the same optimal schedule with the one without any preemption. In this section we will talk about what kind of preemption can be allowed in the TWET problem.

First we give several different schemes to preempt the jobs. For each scheme, we will give the corresponding lemma to show whether such a preemption scheme is allowed in the TWET problem.

Scheme 1. The early jobs can be preempted into pieces and all pieces are early. And the earliness of the preempted early job is defined as the earliness of the last piece, i.e., its due date minus the completion time of the last piece. No tardy job is preempted.

Scheme 2. No early job is preempted. Only tardy jobs can be preempted into several pieces to be completed non-contiguously and all pieces are tardy. The tardiness of the preempted tardy job is defined as the tardiness of the last piece of the job, i.e., the completion time of the last piece minus its due date.

Scheme 3. Both early and tardy jobs can be preempted. All pieces of early jobs are early while all pieces of tardy jobs are tardy. The earliness of the preempted early job is defined as the earliness of the first piece, i.e., its due date minus the completion time of the first piece, while the tardiness of the preempted tardy job is defined as the tardiness of the last piece, i.e., the completion time of the last piece minus its due date.

Lemma 2.3.1 *Scheme 1 is not allowed.*

Proof. Just consider the early jobs in TWET problem. Since the earliness of the preempted early job is defined as the earliness of the last piece, we can cut a small piece an early job and put that small piece near the due date. For all the early jobs, we can cut the smaller and smaller pieces repeatedly to put near the due date (other pieces can be put before the smaller ones) to make the total earliness smaller. For example, suppose we get an optimal schedule $\mathcal{S}$ for the TWET problem with Scheme 1. For an early job, say job J_i with processing time p_i and weight w_i in the optimal schedule, suppose the length of its last piece is l_i, $0 < l_i \leq p_i$. We can cut this last piece into two pieces with one of them has length x, $0 < x < l_i$ and the other has length $l_i - x$. We can put the piece with length x at time 0 after moving all pieces of jobs before the last piece of J_i towards the due date. In such way, the earliness of J_i doesn't change while every early job before it has a smaller earliness since they have been moved towards the due date. Thus, we have a better schedule with smaller objective value. This contradicts the optimality. The Scheme 1 preemption is not allowed. $\square$

Lemma 2.3.2 *Scheme 2 is allowed for the TWET problem.*

Proof. According to Scheme 2, the early jobs can not be preempted. Thus, the early jobs perform the same way with the ones in TWET problem without preemption. Just consider the tardy jobs. First, assume in an optimal schedule S, all preempted pieces of a tardy job are in the same interval. To simplify the proof, we assume that the preempted tardy job is job J_p and it has only two pieces p_1 and p_2, while p_2 is the last piece. Exchange p_1 with the block of tardy jobs between p_1 and p_2. After the exchanging, p_1 and p_2 come together to become a whole. The tardiness of every tardy job between p_1 and p_2 is smaller since they have been moved earlier, while the tardiness of J_p doesn't change. This contradicts the optimality of schedule S. Thus there is no preemption in the optimal schedule. Suppose in an optimal schedule the preempted pieces of a tardy job locate in different intervals. We assume there're only two pieces to simplify the proof, one is in intervals I_i, denoted as piece p_i and the other is in I_j, denoted as piece p_j, $1 \leq i < j \leq K + 1$. Denote the total weight of early jobs between p_i and p_j as W_E and denote the total weight of tardy jobs between p_i and p_j as W_T. If $W_E > W_T$, we can exchange p_j and the whole block between p_i and p_j to get a smaller objective value. Otherwise, we can exchange p_i and the whole block between p_i and p_j to get a smaller objective value. Such exchanging makes the tardy job no preemption. Thus, in the optimal schedule, there's no preemption for the tardy jobs. Scheme 2 is allowed. □

Lemma 2.3.3 *Scheme 3 is allowed.*

Proof. From Lemma 2.3.2, the preemption in Scheme 3 is allowed for the tardy jobs. Then for the early jobs, since the earliness of the preempted early job is calculated as the earliness of the first piece, it is a symmetrical mirroring of the tardiness while the tardiness of the preempted tardy job is calculated as the tardiness of the last piece. We can follow the same proof as in Lemma 2.3.2 to prove that the preemption for the early jobs is also allowed. Thus, Scheme 3 is allowed. □

With the allowed preemption scheme, we can extend the original TWET problem to the new one with some preemption. The new problem provides more flexible structural properties from which we may benefit. We don't use such properties of preemption in our algorithm for the TWET problem with a constant number of distinct due dates and polynomially related weights. Such lemmas may be helpful for the more general TWET problem, and that's why we list them above.

Chapter 3

FPTAS for the Single Machine TWET Problem with Common Due Date

Kellerer and Strusevich [23] proposed an FPTAS for the single machine TWT [22] and TWET [23] problem with a common due date. Our work is based on and inspired by their work. Thus in this chapter, we give a brief introduction of their work on TWT and TWET problem, and show how they use a rounding scheme to get the FPTAS.

3.1 FPTAS for TWT Problem with Common Due Date

3.1.1 Preliminaries

Recall the TWT problem given in Section 1.1. We consider the TWT problem with a common due date in this chapter, that is, all the n jobs have the same due date d. Thus, in a schedule, the tardiness for each job J_j becomes $T_j = d - C_j$, $j = 1, \ldots, n$ and the objective function becomes $\sum_{j=1}^{n} w_j T_j$. We want to find a schedule to minimize the objective value. For this problem, the job which starts before or on the due date and completes after the due date is called a *straddler*. Obviously, for each schedule, there's a specific straddler and the jobs before it are early jobs and the jobs after it are tardy. Kellerer et al. [22] guess the straddler at the beginning through exhaustive enumeration, i.e, taking each job as the possible straddler. For the remaining $m = n - 1$ non-straddling jobs, they first insert these jobs through a dynamic programming process and then insert the straddler at the very end. The m non-straddling jobs can be inserted according to the following:

1. Before inserting the jobs, all the non-straddling jobs are ordered by the WSPT rule.

2. The first early job starts from time zero and the first tardy job starts from the due date.

3. All the early jobs are processed as a block without intermediate idle time.

4. All the tardy jobs are processed as a block in WSPT order without intermediate idle time.

The goal is to find the minimum weighted tardiness of the non-straddling jobs for each chosen straddler.

3.1.2 DP for the Non-straddling Jobs

It has been proved that the tardy jobs are processed in WSPT order (see Lemma 2.2.1) for the m non-straddling jobs. With the objective to minimize the total weighted tardiness for m non-straddling jobs, i.e., $Z_m = \sum_{j=1}^{m} w_j T_j$, it is important to decide which jobs are scheduled as early and which as tardy. Therefore, a Boolean decision variable x_j is defined for each job J_j, where $x_j = 1$ if job J_j is tardy and $x_j = 0$ otherwise. Then the total weighted tardiness for the m non-straddling jobs in a feasible schedule is given by $Z_m = \sum_{j=1}^{m} w_j (\sum_{i=1}^{j-1} p_i x_i) x_j$. Here "feasible" means that the total processing time of all early jobs is no more than d, i.e., $\sum_{j=1}^{m} p_j(1 - x_j) \leq d$.

Define k as the number of jobs scheduled so far; Z_k is the current value of the objective function; $A_k = \sum_{j=1}^{k} p_j$ is the total processing time of all k jobs; $y := \sum_{j=1}^{k} p_j x_j$ is the total processing time of the tardy jobs among the first k jobs; $W := \sum_{j=1}^{k} w_j x_j$ is the total weight of these tardy jobs.

A state of the dynamic programming (DP) is defined as (k, Z_k, y_k, W_k) after the first k jobs have been inserted. The initial state is $(0, Z_0, y_0, W_0) = (0, 0, 0, 0)$. For all k from 0 to $m-1$, the transition from a state (k, Z_k, y_k, W_k) into any state $(k+1, Z_{k+1}, y_{k+1}, W_{k+1})$ is defined as follows:

1. Define $x_{k+1} = 1$. Then the job J_{k+1} in WSPT order is decided to be early feasibly, i.e., $A_k - y_k \leq d$. Then $Z_{k+1} = Z_k$ and $y_{k+1} = y_k$, $W_{k+1} = W_k$.

2. Define $x_{k+1} = 0$, i.e., the job J_{k+1} is processed tardy which is always feasible. Then $y_{k+1} = y_k + p_{k+1}$, $W_{k+1} = W_k + w_{k+1}$ and $Z_{k+1} = Z_k + w_{k+1} y_{k+1}$.

The DP above generates a collection of states (m, Z_m, y_m, W_m) after inserting all m non-straddling jobs. The decision variables can be found by backtracking and we can get the corresponding schedule for each state. Since every transition between the states in the DP process is feasible, every state in the form of (m, Z_m, y_m, W_m) is feasible as well, and corresponds to a real schedule sequence of m jobs. Then the next step is to insert the chosen straddler back to get the final schedule for all n jobs.

3.1.3 Inserting the Straddler Back

Before applying the above DP for the m non-straddling jobs, the straddler has been guessed at the beginning. Then after we get the schedule of m jobs, we need to insert the straddler back into every feasible schedules from the DP. More specifically, the straddler should be processed starting from the completion time of the last early job. The tardy block also needs to be pushed forward to future time to make enough space to insert the straddler. Assume that the straddler has weight w and processing time p. For each schedule S_m, compute $x = A_m - y_m + p - d$; x is the tardiness of the straddler, and it's the amount of processing time by which the tardy block is pushed as well. Then after inserting the straddler, the tardiness of all the tardy jobs in the tardy block is increased by x. From the definition of the straddler, we only consider that $0 < x \leq p$. The total weighted tardiness for the TWT problem with common due date is $Z = Z_m + (w + W_m)x$. For every chosen straddler, we can find a schedule with minimal Z. After considering all possible straddlers, an optimal schedule for the TWT problem with common due date can be found.

Since the schedule S_m with minimum Z_m may not give the minimal Z from the formula $Z = Z_m + (w + W_m)x$, all the schedules S_m after inserting m jobs according to the DP should be considered when inserting the straddler back.

3.1.4 Rounding Scheme

First consider the running time of the DP algorithm in Section 3.1.2. Since each job of the m non-straddling jobs can be scheduled as either early or tardy, the dynamic programming above will generate $O(2^m) = O(2^n)$ states. This complexity is exponential in the problem size and it is too inefficient when the problem becomes large. Then a rounding scheme should be used to reduce the number of states to make the number of generated states polynomial. To reduce the number of states, it's necessary to reduce the number of distinct values for each variable in the state. Meanwhile, we do not want to lose much accuracy. To do this, Kellerer et al [23] propose an FPTAS using a rounding scheme. The complexity of the algorithm is polynomial in the problem size, and at the same time, the solution is bounded by a factor of $1 + \epsilon$ of the optimal. The number $1 + \epsilon$ is the approximation ratio. The rounding scheme is defined as follows.

1. Find an upper bound Z^{UB} on the optimal objective value Z^* such that $Z^{UB}/Z^* \leq 2$.

2. For any $\epsilon > 0$, define $Z_{LB} = \frac{1}{2}Z^{UB}$ and $\delta = \frac{\epsilon Z_{LB}}{4m}$.

3. Sort all the $h \leq m$ distinct weights in decreasing order, i.e., $w_{\pi(1)} > w_{\pi(2)} > \cdots > w_{\pi(h)}$. Split the interval $[0, \frac{z^{UB}}{w_{\pi(h)}}]$ into h intervals $I_1 = [0, \frac{z^{UB}}{w_{\pi(1)}}]$, $I_2 = [\frac{z^{UB}}{w_{\pi(1)}}, \frac{z^{UB}}{w_{\pi(2)}}]$, ..., $I_h = [\frac{z^{UB}}{w_{\pi(h-1)}}, \frac{z^{UB}}{w_{\pi(h)}}]$. Further divide each interval $I_j, 1 \leq j \leq h$ into subintervals I_j^r of length $\frac{\delta}{w_{\pi(j)}}$.

4. Store the initial state $(0, 0, 0, 0)$. For each k, $1 \leq k \leq m$, do the following:

   (a) For each state (k, Z_k, y_k), round Z_k up to the next multiple of δ. Round the value of W_k up to the nearest power of $(1 + \frac{\epsilon}{2})^{\frac{1}{m}}$.

   (b) For the states with the same Z_k, W_k and a subinterval I_j^r, determine the smallest and largest value of y_k that belong to I_j^r as y_k^{min} and y_k^{max}. Store only two states(k, Z_k, y_k^{min}) and (k, Z_k, y_k^{max}).

From the rounded DP with the rounding scheme, the numbers of distinct Z_k, W_k and y_k are rounded to polynomial to guarantee that the total number of states is polynomial. Kellerer et al. have proved that this rounding scheme gives an FPTAS in [22].

3.2 FPTAS for TWET with Common Due Date

3.2.1 Preliminaries

First recall the TWET problem given in Section 1.1. We consider the TWET problem with a common due date in this chapter, that is, all the n jobs have the same due date d. Thus, in a schedule, for each job $J_j, j = 1, 2, \ldots, n$, the earliness becomes $E_j = d - C_j$ and the tardiness becomes $T_j = C_j - d$. The goal is to find a schedule to minimize the objective function $\sum_{j=1}^{n} w_j(E_j + T_j)$. For this problem, the job which starts before the due date d and finishes after d is defined as the *straddler*.

If the total processing time $P = \sum_{j=1}^{n} p_j$ is greater than the common due date d, the due date is said to be *small* or *restrictive*. Otherwise, when $P \leq d$, the due date is called *large* or *nonrestrictive*. Such a classification is needed since the due date may influence the structure of a feasible schedule and complexity status of the problem. Kovalyov and Kubiak [26] present an FPTAS for the TWET problem with the nonrestrictive due date. Kellerer and Strusevich [23] focus on the more difficult restrictive case.

For the single machine TWET problem with restrictive due date, the optimal schedule can be sought for in two classes of schedules as demonstrated in Property 1 of [16]. First, some job will complete exactly at time d in an optimal schedule, i.e., it has neither earliness nor tardiness. There is no intermediate idle time between jobs, but there may be some idle time before the first early job. We call this class of schedules *Class 1*. Second, in an optimal schedule, the early jobs are processed starting at time zero and are followed by the *straddler* which starts before time d but finishes after d; then, the straddler is followed by the tardy jobs. This class of schedules is called *Class 2*. Namely, Class 1 means scheduling without a straddler while Class 2 means scheduling with a straddler. When the due date is nonrestrictive, only Class 1 schedules exist.

For *Class 1*, there is no straddler which starts before time d and is completed after d. Then all the n jobs are non-straddling jobs. For *Class 2*, we can guess the straddler first, and then there are $n-1$ non-straddling jobs left. It's easy to see that any non-straddling job finishing before the due date d is early and starting after d is tardy. Thus, the due date separates all non-straddling jobs into two categories: early and tardy jobs. We can process these jobs according to the following:

1. Before inserting the jobs, all the non-straddling jobs are ordered by the WSPT rule and inserted in the schedule one by one.
2. All early jobs are processed as a block in inverse WSPT (I-WSPT) order without intermediate idle time and complete by the due date.
3. All the tardy jobs are processed as a block in WSPT order without intermediate idle time and start at the due date.

The goal is to find the minimum weighted earliness and tardiness of the non-straddling jobs. And for Class 2 schedules, we need also to insert the straddler back to get the final schedule.

3.2.2 DP for the Non-straddling Jobs

For the non-straddling jobs, it has been proved that the early jobs are processed in inverse WSPT order and the tardy jobs are processed in WSPT order (see Corollary 2.2.1). To minimize the total weighted earliness and tardiness for m non-straddling jobs, i.e., $Z_m = \sum_{j=1}^{m} w_j(E_j + T_j)$, it is important to decide which jobs are scheduled as early and which as tardy. Therefore, a Boolean decision variable x_j is defined for each job J_j, where $x_j = 1$ if job J_j is early and $x_j = 0$ otherwise. Then the total weighted earliness and tardiness for the m non-straddling jobs in a feasible schedule is given by $Z_m = \sum_{j=1}^{m} w_j(\sum_{i=1}^{j-1} p_i x_i)x_j + \sum_{j=1}^{m} w_j(\sum_{i=1}^{j} p_i(1-x_i))(1-x_j)$, which can be written as

$$Z_m = \sum_{1 \le i < j \le m} p_i w_j x_i x_j + \sum_{1 \le i < j \le m} p_i w_j (1-x_i)(1-x_j) + \sum_{j=1}^{m} p_j w_j (1-x_j)$$

Here "feasible" means that the total processing time of all early jobs is no more than d, i.e., $\sum_{j=1}^{m} p_j x_j \le d$. Thus, finding a best schedule for the non-straddling m jobs reduces to the following Boolean quadratic programming problem:

$$\text{Minimize} \quad Z_m = \sum_{1\le i<j\le m} p_i w_j x_i x_j + \sum_{1\le i<j\le m} p_i w_j (1-x_i)(1-x_j) + \sum_{j=1}^{m} p_j w_j (1-x_j) \tag{3.2.1}$$

$$\text{Subject to} \quad \sum_{j=1}^{m} p_j x_j \le d$$

$$x_j \in 0,1, j = 1,2,\ldots,m.$$

This is a special case of a symmetric quadratic knapsack problem and we will show to solve a symmetric quadratic knapsack problem in the next section.

3.2.3 Symmetric Quadratic Knapsack Problem

The *Symmetric Quadratic Knapsack Problem*, or SQKP, is defined as follows:

$$\text{Minimize} \quad Z_m = \sum_{1\le i<j\le m} \alpha_i \beta_j x_i x_j + \sum_{1\le i<j\le m} \alpha_i \beta_j (1-x_i)(1-x_j) + \sum_{j=1}^{m} p_j w_j (1-x_j) \tag{3.2.2}$$

$$+ \sum_{j=1}^{m} \mu_j x_j + \sum_{j=1}^{m} \nu_j (1-x_j) + \Gamma$$

$$\text{Subject to} \quad \sum_{j=1}^{m} \alpha_j x_j \le A$$

$$x_j \in 0,1, j = 1,2,\ldots,m.$$

where m is the number of items, all coefficients α_j, β_j, μ_j, ν_j, $j = 1,2,\ldots,m$, and Γ are non-negative integers.

Because both the linear and quadratic parts of the objective function are divided into two parts, one depending on x_j, and the other depending on $(1-x_j)$, this problem is called *symmetric*. Notice that the coefficients α_j in the linear constraint are the same as in the quadratic terms of the objective function. We can view α_j as the weight of item j, $1 \le j \le m$, i.e., $x_j = 1$

means that item j is put into the knapsack, while $x_j = 0$ means that this item is not placed into the knapsack. The value A is the capacity of the knapsack.

In [24], a dynamic programming (DP) algorithm has been presented for the SQKP. The items are scanned in their numbering order and the corresponding decision variables x_j, $1 \leq j \leq m$ are given either the value of 1 (the item is put into the knapsack) or 0 (the item isn't put into the knapsack). Define k as the number of items processed so far; Z_k is the current value of the objective function; $y := \sum_{j=1}^{k} \alpha_j x_j$ is the total weight of the items put into the knapsack. Also compute the values $A_k = \sum_{j=1}^{k} \alpha_j, k = 1, 2, \ldots, m$ which is the total weight of all the k jobs.

The state of the dynamic programming is defined as (k, Z_k, y_k) after the first k items have been placed. The initial state is $(0, Z_0, y_0) = (0, 0, 0)$. For all k from 0 to $m - 1$, the transition from a state (k, Z_k, y_k) into any state $(k + 1, Z_{k+1}, y_{k+1})$ by assigning the next variable x_{k+1} is defined as follows:

1. Define $x_{k+1} = 1$, i.e., the item $k + 1$ can be placed into the knapsack feasibly, i.e., $y_k + \alpha_{k+1} \leq A$. Then $Z_{k+1} = Z_k + \beta_{k+1} y_k + \mu_{k+1}$ and $y_{k+1} = y_k + \alpha_{k+1}$.

2. Define $x_{k+1} = 0$, i.e., the item $k+1$ is not put into the knapsack which is always feasible. Then $Z_{k+1} = Z_k + \beta_{k+1}(A_k - y_k) + \nu_{k+1}$ and $y_{k+1} = y_k$.

The DP above outputs a collection of states (m, Z_m, y_m) after assigning all m items. The decision variables can be found by backtracking and we can get the corresponding schedule for each state. Similarly, a dual DP is needed with the state form of $(k, Z_k, \tilde{y_k})$ to design an FPTAS, where k and Z_k have the same meaning as above, while $\tilde{y_k} = A_k - y_k$, which is the total weight of the items not put into the knapsack. This dual DP follows the similar transition between states as the above DP. In the next section we show a rounding scheme applied to the DP to keep a polynomial number of states.

3.2.4 Rounding Scheme

Similarly to the DP in Section 3.1.4, the dynamic programming above also generates $O(2^m) = O(2^n)$ states. We need to use the rounding scheme to reduce the number of states. The number of distinct values for each variable in the state should be reduced while no much accuracy is lost. Kellerer et al. [23] propose an FPTAS using a rounding scheme. The complexity of the algorithm is polynomial, and the solution is bounded by the approximation ratio $1+\epsilon$ of the optimal. The rounding scheme works as follows:

1. Find an upper bound Z^{UB} on the optimal objective value Z^* such that $Z^{UB}/Z^* \leq \rho$. For any $\epsilon > 0$, define $Z_{LB} = \frac{1}{\rho}Z^{UB}$ and $\delta = \frac{\epsilon Z_{LB}}{2m}$.

2. Sort all the $h \leq m$ distinct weights in decreasing order, i.e., $w_{\pi(1)} > w_{\pi(2)} > \cdots > w_{\pi(h)}$. Split the interval $[0, \frac{z^{UB}}{w_{\pi(h)}}]$ into h intervals $I_1 = [0, \frac{z^{UB}}{w_{\pi(1)}}]$, $I_2 = [\frac{z^{UB}}{w_{\pi(1)}}, \frac{z^{UB}}{w_{\pi(2)}}]$, ..., $I_h = [\frac{z^{UB}}{w_{\pi(h-1)}}, \frac{z^{UB}}{w_{\pi(h)}}]$. Further divide each interval $I_j, 1 \leq j \leq h$ into subintervals I_j^r of length $\frac{\delta}{w_{\pi(j)}}$.

3. For each k, $1 \leq k \leq m$, do the following:

   (a) For each state (k, Z_k, y_k), round Z_k up to the next multiple of δ. For the states with the same Z_k and a subinterval I_j^r, determine the largest and smallest values of y_k and store only two states(k, Z_k, y_k^{min}) and (k, Z_k, y_k^{max}).

   (b) For each dual state $(k, Z_k, \tilde{y_k})$, round Z_k up to the next multiple of δ. Similarly, for the states with the same Z_k and a subinterval I_j^r, store only two states$(k, Z_k, \tilde{y_k}^{min})$ and $(k, Z_k, \tilde{y_k}^{max})$.

   (c) For each state (k, Z_k, y_k) stored in Step 3(a), store the dual state $(k, Z_k, \tilde{y_k})$ additionally where $\tilde{y_k} = A_k - y_k$. Similarly, for each dual state $(k, Z_k, \tilde{y_k})$ stored in Step 3(b), store the state (k, Z_k, y_k) additionally where $y_k = A_k - \tilde{y_k}$.

According to the rounding scheme, distinct Z_k, y_k and $\tilde{y_k}$ are rounded to a polynomial number to guarantee that the total number of states is polynomial. In [23], Kellerer et al. have proved that this rounding scheme gives an FPTAS when ρ is a constant.

Compare the TWET problem with a common due date in formula (3.2.1) and the problem SQKP in formula (3.2.2). We observe that the former problem is a special case of the latter with:

$$\alpha = p_j,\ \beta = w_j,\ \mu_j = 0,\ \nu_j = w_j p_j,\ j = 1, 2, \ldots, n,\ A = d,\ \Gamma = 0.$$

Thus, we can use the above DP and rounding scheme to solve the TWET problem. Kellerer et al. [23] also have shown a constant-ratio approximation algorithm. Here we skip that part and details can be found in section 4 of [23]. The above rounding scheme gives an FPTAS for the TWET problem with a common due date.

3.2.5 Scheduling Without a Straddler

For Class 1 schedule defined in Section 3.1.1, since there's no straddler, all n jobs are non-straddling jobs. It's easy to see that the above DP with $m = n$ can be used directly to get the optimal solution which corresponds to the smallest value of Z_m among all found states of the form (m, Z_m, y_m). Moreover, applying the DP with the rounding scheme above, an FPTAS can be developed to get an approximation solution with a ratio $1 + \epsilon$.

3.2.6 Scheduling With a Straddler

Recall that we have guessed a straddler for Class 2 schedule in Section 3.2.1. Through exhaustive enumeration, we select every job from all the n jobs as a possible straddler. More specifically, the straddler is processed before the due date d and is completed after d. Besides, there's no idle time before the first early job after inserting the straddler. Suppose the straddler has weight w and processing time p. Then after inserting the remaining $m = n - 1$ according to the DP in Section 3.2.2, the cases either $y_m = d$ or $y_m + p \leq d$ must be ignored since the chosen job cannot be inserted as a straddler. Compute $x = \frac{d - y_m}{p}$ and we only need to consider $0 < x < 1$. Let W denote the total weight of all n jobs. For the schedule corresponding to the state (n, Z_m, y_m), define W_m to be the total weight of the early jobs. We can compute the total weighted earliness and tardiness for the TWET problem with common due date as $Z = Z_m + W_m px + (W - W_m)p(1 - x)$.

Thus, we modify the state of the DP algorithm in Section 3.2.2 into the form (k, Z_k, y_k, V_k) where k, Z_k and y_k have the same meaning as in the DP algorithm, while

$$V_k := p \sum_{j=1}^{k} w_j x_j$$

denotes the total weight of the early jobs times the processing time of the straddler. Then the transition from state (k, Z_k, y_k, V_k) to $(k+1, Z_{k+1}, y_{k+1}, V_{k+1})$ becomes

1. If job J_{k+1} is early when $y_k + p_{k+1} \leq d$,

$$Z_{k+1} = Z_k + w_{k+1} y_k,\ y_{k+1} = y_k + p_{k+1},\ V_{k+1} = V_k + w_{k+1} p$$

2. If job J_{k+1} is tardy,

$$Z_{k+1} = Z_k + w_{k+1}(A_k - y_k) + w_{k+1} p_{k+1},\ y_{k+1} = y_k,\ V_{k+1} = V_k$$

Obviously, with the final states of form (m, Z_m, y_m, V_m), the total objective value is $Z = Z_m + (2V_m - Wp)x + (Wp - V_m)$ where W and x is defined above.

Since the state in DP has changed into the new one (k, Z_k, y_k, V_k) from (k, Z_k, y_k), a new rounding scheme is needed to keep the total number of states polynomial. We modify the Step 3(a) in the rounding scheme in Section 3.2.4 as follows: each time round up Z_k and V_k for all states to the next multiple of δ.

For the states having the same value of Z_k, W_k and a subinterval I_j^r, determine the largest and smallest values of y_k such that $y_k \in [y_k^{min}, y_k^{max}]$, then save only two states (k, Z_k, y_k^{min}, V_k) and (k, Z_k, y_k^{max}, V_k). Step 3(b) is altered similarly. With the modified rounding scheme, an FPTAS can be obtained.

Combining the Class 1 with Class 2 schedules, the solution for the TWET problem on a single machine with a common due date is the smaller one between the solutions of the two kinds of schedules. Thus overall, an FPTAS has been presented to solve the problem.

Chapter 4

FPTAS for the Single Machine TWET Problem with Distinct Due Dates and Polynomially Related Weights

Based on the work of TWET problem on the single machine with common due date [23] and the related TWT problem with common due date [22] and constant distinct due dates [21], we analyze the structural properties of an optimal schedule of the TWET problem with a constant number of distinct due dates and polynomially related weights to define a schedule in which only the non-straddling jobs are considered. We propose a dynamic programming algorithm to obtain the abstract schedule and convert it into an FPTAS via a rounding scheme.

4.1 Preliminaries

Recall the TWET problem with the Assumption 1.1.1 in Section 1.1. The number K of distinct due dates with $d_1 < d_2 < \cdots < d_K$ is a constant. And the maximal weight w_{max} and the minimal weight w_{min} of all the n jobs is polynomially related, i.e., $\frac{w_{max}}{w_{min}} = O(poly(n))$.

For convenience, we also define the artificial due date $d_0 = 0$. Then the distinct due dates partition the time horizon into $K+1$ intervals $I_l = [d_{l-1}, d_l)$ for $l = 1, ..., K$, and $I_{K+1} = [d_K, \infty)$. Besides, we group the jobs into K classes $C_1, C_2, ..., C_K$ according to their due dates.

In any schedule of the n jobs, a job that finishes before or on its due date is an *early* job, otherwise it is *tardy*. We also call any job that (i) ends at a due date, or (ii) starts before a due date but finishes after it, a *straddler*. Note that here we give a broader definition of the *straddler* when we treat the job that ends at a due date as a straddler. In order to simplify the exposition in this thesis we will assume that these straddlers are *distinct*, i.e., no straddler straddles more than one due date. Thus we have K straddlers for the K due dates. In what follows, we will assume that we have *guessed* the straddlers $s_1, ..., s_K$ for the K due dates. By "guessing" we mean an exhaustive enumeration of all $O(n^K)$ possibilities, solving the problem for each, and outputting the best of these solutions. Then the problem can be divided into two parts: first calculate an optimal schedule for the remaining $m = n - K$ jobs, and then insert the K straddlers.

Without loss of generality, we assume that all values are integers, and the jobs are ordered according to their Inverse Weighted Shortest Processing Time (I-WSPT) rule, i.e., $\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \ldots \geq \frac{p_n}{w_n}$. We will also assume that we have guessed an upper bound Z^{ub} such that for the optimal value OPT we have $Z^{ub}/2 \leq OPT \leq Z^{ub}$. This can be done by running the algorithm with $Z^{ub} = 2^x$, for all $x = 0, 1, \ldots, U$, with 2^U being a trivial upper bound of OPT, e.g. $U = \log(n^2 w_{max} p_{max}) = O(\log n + \log w_{max} + \log p_{max})$.

For the remaining $m = n - K$ jobs, we start to schedule them one by one according to the I-WSPT order. To insert a job in some interval, we need to check if there's enough space to insert it. Besides, we still need to check that we can insert the straddler back in the final step to get a feasible schedule. Thus, we come up with several feasibility checking conditions to make sure

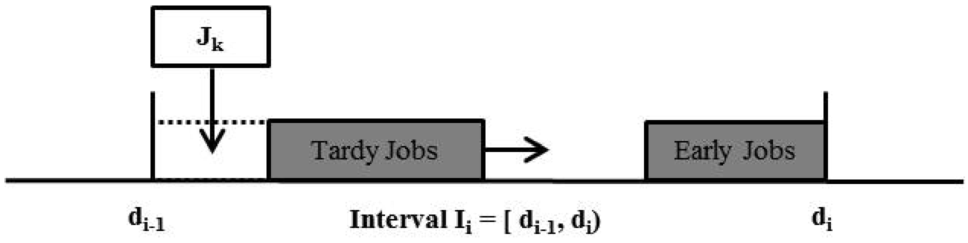

Figure 4.1: Insert job J_k as a tardy job in interval I_i

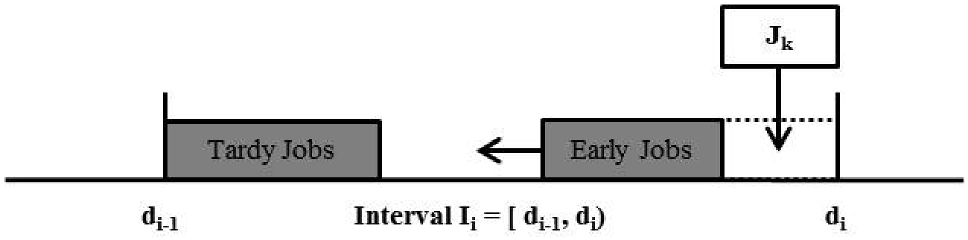

Figure 4.2: Insert job J_k as an early job in interval I_i

that the schedule will be feasible. After checking such conditions, we insert the jobs from each due date, i.e, every time moving the existing tardy jobs block forward to make the space for the new tardy job as shown in Figure 4.1, or moving the existing early jobs block backward to make the space for the new early job as Figure 4.2 shows. The following are structural properties of an optimal schedule with an extension for K due dates from Corollary 2.2.1, proven by simple exchange arguments:

Lemma 4.1.1 *In any interval $I_j, j = 1, .., K+1$, the early jobs must be processed in Inverse WSPT (I-WSPT) order and the tardy jobs must be processed in WSPT order.*

Lemma 4.1.2 *In any interval $I_j, j = 1, .., K+1$, the tardy jobs start right after straddler s_{j-1}, and they are processed contiguously, followed possibly by idle time, and then followed by the early jobs which are processed contiguously and end right before s_j.*

Since the algorithm will construct a schedule by inserting the jobs one-by-one, for every partial schedule we define the following variables:

- $y_k^i,\ 1 \le i \le K+1,\ 1 \le k \le m$: the total processing time of those (tardy) jobs among the first k jobs, that are processed in I_i. Also $y_k^1 = 0$.
- $W_k^i,\ 1 \le i \le K+1,\ 1 \le k \le m$: the total weight of those (tardy) jobs among the first k jobs, that are processed in I_i. Also $W_k^1 = 0$.
- $e_k^i,\ 1 \le i \le K+1,\ 1 \le k \le m$: the total processing time of those (early) jobs among the first k jobs, that are processed in I_i. Also $e_k^{K+1} = 0$.
- $V_k^i,\ 1 \le i \le K+1,\ 1 \le k \le m$: the total weight of those (early) jobs among the first k jobs, that are processed in I_i. Also $V_k^{K+1} = 0$.
- $A_k^t,\ 1 \le t \le K,\ 1 \le k \le m$: the total processing time of the class C_t jobs among the first k jobs. Notice that these quantities can be calculated in advance.

We call the schedule for these m jobs an *abstract* schedule to distinguish the final schedule for all the n jobs. In the next section we will show how to develop a dynamic programming algorithm to get a schedule.

4.2 An Exact Dynamic Programming Algorithm

In this section we present a dynamic programming (DP) algorithm that calculates an optimal schedule. As mentioned above, we will leave the insertion of the K straddlers at the very end. Let $m = n - K$ be the number of the rest of the jobs. The states of the DP are organized in stages, with stage k corresponding to the insertion of the first k jobs in I-WSPT order in the schedule. The transitions from states in stage k to ones in stage $k+1$ correspond to the placement of job J_{k+1} in all possible $K+1$ intervals (either as early or as tardy, depending on the class of J_{k+1}). The method of inserting J_{k+1} in interval I_l is important: If J_{k+1} is tardy in this interval, the job is inserted starting *exactly at* due date d_{l-1}, after we push the other tardy jobs towards the future in order to make room for p_{k+1} time units; if J_{k+1} is early in this interval, the job is inserted ending *exactly at* due date d_l, after we push the other early jobs towards the past in order to make room for p_{k+1} time units. A state in stage k of the DP stores the following tuple:

$$(k, Z_k; V_k^1; y_k^2, W_k^2, e_k^2, V_k^2; y_k^3, W_k^3, e_k^3, V_k^3; \ldots; y_k^{K+1}, W_k^{K+1}, e_k^{K+1}, V_k^{K+1})$$

where Z_k is the total weighted earliness-tardiness of the first k scheduled jobs. Note that some of the $y_k^j, W_k^j, e_k^j, V_k^j$ may not exist and we don't keep e_k^1. In stage 0 there is only the state $(0, 0, \ldots, 0)$. When inserting job J_{k+1} in an interval, and in order for this transition to be feasible, there must be at least p_{k+1} free space in the interval. Hence, in order to be able to check of the feasibility of job placements, we define $L_k^{(j-1)j}$ to be the free space in interval $I_j,\ 1 \le j \le K+1$ in stage $0 \le k \le m$. Then, we have

$$L_k^{(j-1)j} = \begin{cases} d_1 - \sum_{t=1}^{K} A_k^t + \sum_{j=2}^{K+1} (y_k^j + e_k^j), & j = 1 \\ d_j - d_{j-1} - (y_k^j + e_k^j), & 2 \le j \le K \\ \infty, & j = K+1 \end{cases} \tag{4.2.1}$$

Let $L_k^{0j} = \sum_{l=1}^{j} L_k^{(l-1)l}$ be the free space from 0 to d_j. We will also need the following quantities:

$$u_k^i = \begin{cases} 0, & i = 0 \\ \max\{0, p_{s_i} - L_k^{(i-1)i} + u_k^{i-1}\}, & 1 \le i \le K-1 \end{cases} \tag{4.2.2}$$

In order to check the feasibility of inserting the $(k+1)$-th job J_{k+1} of class C_t in interval $I_j = [d_{j-1}, d_j)$, we check the following two conditions:

Condition 1 Check whether $L_k^{(j-1)j} \ge p_{k+1}$ holds.

Condition 2 After inserting job J_{k+1}, check whether $u_{k+1}^i \le L_{k+1}^{i(i+1)}$, for all $1 \le i \le K-1$.

Condition 1 is used to check whether there is enough space to insert job J_{k+1} in interval I_j. We will show below that the fulfillment of Condition 2 implies that there is still enough empty space for the straddlers after inserting J_{k+1}.

If the placement of J_{k+1} in I_j satisfies these two conditions, then it defines a feasible transition from state $(k, Z_k; \ldots)$ to state $(k+1, Z_{k+1}; \ldots)$. The latter state must satisfy the following:

- If J_{k+1} is *early*, i.e., $j \leq t \leq K$, then

  $$Z_{k+1} = Z_k + V_k^j p_{k+1},\ e_{k+1}^j = e_k^j + p_{k+1},\ V_{k+1}^j = V_k^j + w_{k+1},$$

  and all the other variables remain unchanged.

- If J_{k+1} is *tardy*, i.e., $1 \leq t \leq (j-1)$, then

  $$Z_{k+1} = Z_k + (W_k^j + w_k)p_{k+1},\ y_{k+1}^j = y_k^j + p_{k+1},\ W_{k+1}^j = W_k^j + w_{k+1},$$

  and all the other variables remain unchanged.

If $Z_{k+1} > Z^{ub}$, then the transition is rendered infeasible (although it satisfies Conditions 1,2). If at some point we determine that this inequality is true for all possible insertions of J_{k+1} then we reject Z^{ub}, we replace it with a new $Z^{ub} := 2Z^{ub}$, and start the algorithm from scratch.

Placement of the straddlers

After calculating the states of stage m, we insert the K straddlers. For straddler s_j, let x_j be its part that is executed before d_j. Then, given a final state $(m, Z_m; \ldots)$, we solve the following linear program:

$$\min \sum_{j=1}^{K} (V_m^j x_j + W_m^{j+1}(p_{s_j} - x_j)) \quad \text{s.t.} \tag{LP}$$

$$\begin{aligned} p_{s_j} - x_j + x_{j+1} &\leq L_m^{j(j+1)} && 1 \leq j \leq K-1 \\ x_1 &\leq L_m^{01} && \\ 0 \leq x_j &\leq p_{s_j} && 1 \leq j \leq K \end{aligned}$$

Lemma 4.2.1 (LP) *above has an optimal solution.*

Proof. The objective is lower bounded by 0. We show that $x_j := p_{s_j} - u_m^j$ is a feasible solution.

Obviously $0 \leq x_j \leq p_{s_j}$. By the definition of u_m^j we have that

$$x_j = \min\{p_{s_j}, L_m^{(j-1)j} - u_m^{j-1}\}, \quad 2 \leq j \leq K-1$$

Therefore $x_j \leq L_m^{(j-1)j} - p_{s_{j-1}} + x_{j-1}$, which implies that $p_{s_j} - x_j + x_{j+1} \leq L_m^{j(j+1)}$, for $1 \leq j \leq K-1$. Also, $x_1 = \min\{p_{s_1}, L_m^{01}\} \leq L_m^{01}$ holds.
Every feasible and bounded minimization problem has an optimal solution. □

4.3 The FPTAS

It is obvious that the pseudo-polynomial algorithm of Section 4.2 computes an optimal schedule. In this section we will produce an FPTAS by rounding the states of the DP part of that algorithm (while we will use the same method for inserting the straddlers at the end).

4.3.1 The Algorithm with Rounding Scheme

Let $\varepsilon > 0$ be the approximation parameter of the FPTAS. We define $Z_{lb} := Z^{ub}/2$ and $\delta = \frac{\varepsilon Z_{lb}}{4m}$. Let there be $N \leq m$ distinct values among w_j, $j = 1, 2, ..., m$; we sort them in decreasing order $w_{\pi(1)} > w_{\pi(2)} > ... > w_{\pi(N)}$. We split the interval $[0, Z^{ub}/w_{\pi(N)}]$ into $x = \lceil \frac{Z^{ub}}{\hat{\delta} w_{\pi(N)}} \rceil$ subintervals $\{H_i\}_{i=1}^{x}$ of length $\hat{\delta} = \frac{\delta}{n^2 w_{max}}$ (note that the length of the last subinterval may be smaller than $\hat{\delta}$).

We change the original problem $\mathcal{P}$ into a new problem $\mathcal{P}'$ with exactly the same set of jobs, but with new due dates $d'_j, j = 1, ..., K$ defined as follows:

$$d'_j = d_j + (j-1)2n\hat{\delta}, \ j = 1, \dots, K \qquad (4.3.3)$$

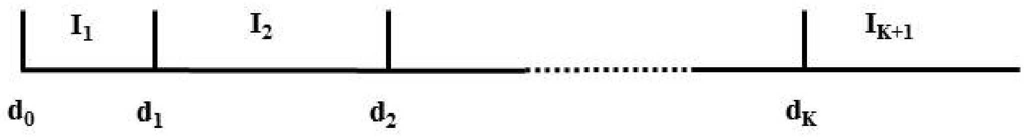

Figure 4.3: The original due dates and intervals

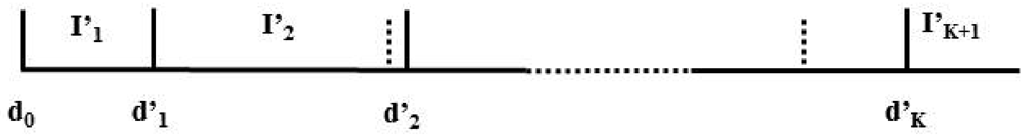

Figure 4.4: The new due dates and intervals

Then we have new intervals $I'_j, j = 1, ..., K+1$. The original due dates and the new due dates are shown in Figures 4.3 and 4.4.

Similarly, we can define the free space L' in each interval, the values u', and the conditions for feasible transitions exactly in the same way as in Section 4.2. We will refer to the latter as Conditions 1' and 2'. We will also need to compare states S produced by the FPTAS to states S^* produced by the exact DP; we will use the asterisk $(^*)$ to denote quantities that belong to S^*, in order to distinguish them from quantities that belong to S.

The new DP algorithm with rounding scheme is given in Figure 4.5.

Note the Step 3 in the Algorithm FPTAS, for the states with the same signature, we keep the only one with the maximal $\sum_{j=2}^{K+1}(y_{k+1}^j + e_{k+1}^j)$. The reason behinds this is that we want to make a bigger free space in the first interval since we don't extend the due date according to equation 4.3.3. We know that all the processing time of the k processed jobs is a specific number. For the state with maximal $\sum_{j=2}^{K+1}(y_{k+1}^j + e_{k+1}^j)$ which is the total processing time of jobs processed between the second interval to the last one among k jobs, the state has the minimum earliness in the first interval which corresponds to a bigger free space in the first interval.

Algorithm FPTAS

Step 1. Order all m jobs in I-WSPT order. The initial state is $(0, 0, ..., 0)$.

Step 2. Each state in stage $1 \leq k < m$

$$(k, Z_k; y_k^1, W_k^1, V_k^1; y_k^2, W_k^2, e_k^2, V_k^2; \dots; y_k^{K+1}, W_k^{K+1}, e_k^{K+1}, V_k^{K+1})$$

will generate at most $K+1$ new states (feasible transitions), one for each placement of job J_{k+1} to interval $I_j,\ j = 1, \dots, K+1$. In these new states, we round Z_{k+1} up to the next multiple of δ (hence Z_{k+1} takes at most $\frac{Z^{ub}}{\delta} = O(\frac{n}{\epsilon})$ distinct values), and W_{k+1}^j and V_{k+1}^j up to the nearest power of $(1 + \frac{\epsilon}{2m})^{1/m}$ (hence W_{k+1}^j and V_{k+1}^j takes $O(n^2 \log W/\varepsilon)$ distinct values, where W is the total weight of the m jobs).

Step 3. For each state produced in Step 2, the values y_{k+1}^1, y_{k+1}^2, $e_{k+1}^2, \dots,$ y_{k+1}^{K+1}, e_{k+1}^{K+1} fall in subintervals H_i; call the tuple of these subintervals the *signature* of the state. We group together the states that have *the same values* of $Z_{k+1}, W_{k+1}^1, V_{k+1}^1, \dots, W_{k+1}^{K+1}, V_{k+1}^{K+1}$ and *the same signature*. From each group, we keep only the state that maximizes $\sum_{j=2}^{K+1}(y_{k+1}^j + e_{k+1}^j)$; all the transitions going to the group states we discard become transitions to the state we keep.

Figure 4.5: Algorithm FPTAS

4.3.2 Proof of Near Optimality

With the above rounding scheme, we are going to show that an optimal schedule is included among the generated schedules and a schedule (may be different from the optimal one) which produces an objective value bounded by $1 + \epsilon$ approximation ratio is also included.

Lemma 4.3.1 *Let $\{S_k^*\}_{k=0}^m$ be the sequence of states produced by the optimal placement of jobs in the exact DP of Section 4.2. Then the same sequence of placements is feasible for Algorithm FPTAS, and the sequence of states $\{S_k\}_{k=0}^m$*

produced satisfies

$$Z_k \leq Z_k^* + \frac{k\epsilon}{2m} Z_k^* + k\delta,\ \forall k. \tag{4.3.4}$$

Proof. We use induction. For $0 \leq k \leq m$, the inductive hypothesis for step k, in addition to (4.3.4), contains the following:

$$\sum_{j=2}^{K+1} (y_k^j + e_k^j) \geq \sum_{j=2}^{K+1} (y_k^{j*} + e_k^{j*}) \tag{I}$$

$$(y_k^j + e_k^j) - (y_k^{j*} + e_k^{j*}) \leq 2k\hat{\delta}, \quad 2 \leq j \leq K+1 \tag{II}$$

$$W_k^{j*} \leq W_k^j \leq W_k^{j*}(1 + \frac{\epsilon}{2m})^{\frac{k}{m}}, \quad 1 \leq j \leq K+1 \tag{III}$$

$$V_k^{j*} \leq V_k^j \leq V_k^{j*}(1 + \frac{\epsilon}{2m})^{\frac{k}{m}}, \quad 1 \leq j \leq K+1 \tag{IV}$$

The placement of job J_{k+1} going from S_k^* to S_{k+1}^* is also feasible going from S_k to S_{k+1}. (V)

where y_k^j, e_k^j, W_k^j, V_k^j are from the state S_k of $\mathcal{P}'$, and y_k^j, e_k^j, W_k^{j*}, V_k^{j*} are from the state S_k^* of $\mathcal{P}$ as described in the lemma.

In order to avoid cluttering our notation, below we use L, u instead of L', u' for problem $\mathcal{P}'$, and use L^*, u^* instead of L, u stand for the corresponding variables for problem $\mathcal{P}$.

For the base case ($k = 0$), first notice that (4.3.4), (I), (II), (III), (IV), (V) hold trivially.

1. First we prove that (I), (II) and (V). Suppose that the optimal sequence places the first job J_1 in interval I_t. We first observe that Conditions 1, 2 must hold for S_0^*:

$$L_0^{(t-1)t*} \geq p_1 \tag{4.3.5}$$

$$L_1^{j(j+1)*} \geq u_1^{j*}, \forall\ j = 1, ..., K-1 \tag{4.3.6}$$

From the definition of L_0 we can easily prove that

$$L_0^{(j-1)j} \geq L_0^{(j-1)j*}, j = 1, ..., K+1 \tag{4.3.7}$$

and therefore

$$L_0^{(t-1)t} \geq L_0^{(t-1)t*} \overset{(4.3.5)}{\geq} p_1$$

and Condition 1' holds. To check whether Condition 2' holds, first note that since only J_1 has been inserted, there's no rounding happening in Step 3 of Algorithm FPTAS after inserting J_1. Thus (4.3.7) implies

$$L_1^{(j-1)j} \geq L_1^{(j-1)j*},\ j = 1, ..., K+1. \tag{4.3.8}$$

By definition, $u_1^0 = 0 = u_1^{0*}$, and the repeated application of (4.3.8) gives

$$u_1^j \leq u_1^{j*}, \forall\ j = 1, ..., K-1. \tag{4.3.9}$$

Combining (4.3.6), (4.3.8) and (4.3.9),

$$L_1^{j(j+1)} \geq L_1^{j(j+1)*} \geq u_1^{j*} \geq u_1^j,\ j = 1, ..., K-1$$

and Condition 2' holds, as well. Hence it is feasible to insert J_1 in I'_t.

We assume that the inductive hypothesis is true up to the placement of job $J_k,\ 1 \leq k \leq m-1$.

For inductive step $k+1$, suppose the optimal DP sequence places J_{k+1} in interval I_t. Conditions 1 & 2 imply

$$L_k^{(t-1)t*} \geq p_{k+1} \tag{4.3.10}$$

$$L_{k+1}^{j(j+1)*} \geq u_{k+1}^{j*},\ j = 1, ..., K-1. \tag{4.3.11}$$

Then (since also $L_k^{K(K+1)} = \infty = L_k^{K(K+1)*}$)

$$L_k^{(j-1)j} \overset{(I)}{\geq} L_k^{(j-1)j*}, j = 1, ..., K+1 \tag{4.3.12}$$

(4.3.10) and (4.3.12) imply that Condition 1' holds.

Let ${y'}_{k+1}^j$ and ${e'}_{k+1}^j$ be the tardiness and earliness after inserting J_{k+1} in I'_t but before the rounding according to Step 3 of the algorithm. From Step 2, we know that

$${y'}_{k+1}^t + {e'}_{k+1}^t = y_k^t + e_k^t + p_{k+1}$$

$$y'^{j}_{k+1} + e'^{j}_{k+1} = y^j_k + e^j_k,\ j = 2, ..., K+1\ \&\ j \neq t$$
$$\sum_{j=2}^{K+1}(y'^{j}_{k+1} + e'^{j}_{k+1}) = \sum_{j=2}^{K+1}(y^j_k + e^j_k) + p_{k+1}$$

and similarly for the optimal DP. Therefore

$$\sum_{j=2}^{K+1}(y'^{j}_{k+1} + e'^{j}_{k+1}) - \sum_{j=2}^{K+1}(y^{j*}_{k+1} + e^{j*}_{k+1}) = \sum_{j=2}^{K+1}(y^j_k + e^j_k) - \sum_{j=2}^{K+1}(y^{j*}_k + e^{j*}_k) \overset{\text{(I)}}{\geq} 0 \tag{4.3.13}$$

$$(y'^{j}_{k+1} + e'^{j}_{k+1}) - (y^{j*}_{k+1} + e^{j*}_{k+1}) = (y^j_k + e^j_k) - (y^{j*}_k + e^{j*}_k) \overset{\text{(II)}}{\leq} 2k\hat{\delta} \tag{4.3.14}$$

Suppose the state S'_{k+1} with values y'^{j}_{k+1} and e'^{j}_{k+1} produced in Step 2 has signature $(H^1, H^2, ...)$. According to Step 3, of all the states with the same signature, we keep only state S_{k+1} with the maximum $\sum_{j=2}^{K+1}(y'^{j}_{k+1} + e'^{j}_{k+1})$. Let the tardiness and earliness of S_{k+1} be y^j_{k+1} and $e^j_{k+1}, \forall j$. Then

$$\sum_{j=2}^{K+1}(y^j_{k+1} + e^j_{k+1}) \geq \sum_{j=2}^{K+1}(y'^{j}_{k+1} + e'^{j}_{k+1}) \overset{(4.3.13)}{\geq} \sum_{j=2}^{K+1}(y^{j*}_{k+1} + e^{j*}_{k+1}). \tag{4.3.15}$$

Since state $S_{k+1}$with y^j_{k+1}, e^j_{k+1} and state S'_{k+1} with $y'^{j}_{k+1}, e'^{j}_{k+1}$ have the same signature $(H^1, H^2, ...)$ and each H^i interval has length $\hat{\delta}$, we have

$$|y^j_{k+1} - y'^{j}_{k+1}| \leq \hat{\delta}$$
$$|e^j_{k+1} - e'^{j}_{k+1}| \leq \hat{\delta}$$

and, therefore,

$$\begin{aligned}(y^j_{k+1} + e^j_{k+1}) - (y^{j*}_{k+1} + e^{j*}_{k+1}) &\leq (y'^{j}_{k+1} + e'^{j}_{k+1}) - (y^{j*}_{k+1} + e^{j*}_{k+1}) + 2\hat{\delta} \\ &\overset{(4.3.14)}{\leq} 2(k+1)\hat{\delta}\end{aligned}$$

Up to now, we have proved that the two inequalities (I),(II) hold after inserting J_{k+1}.

From the definitions and (4.3.15) it is easy to see that

$$L^{(j-1)j}_{k+1} \geq L^{(j-1)j*}_{k+1}, \quad j = 1, ..., K+1 \tag{4.3.16}$$

and this, in turn, implies that

$$u_{k+1}^{j} \leq u_{k+1}^{j*}, \forall\ j = 1, ..., K-1 \tag{4.3.17}$$

(4.3.11),(4.3.16),(4.3.17) show that

$$L_{k+1}^{j(j+1)} \geq L_{k+1}^{j(j+1)*} \geq u_{k+1}^{j*} \geq u_{k+1}^{j}, \forall\ j = 1, ..., K-1$$

and Condition 2' also holds. Hence inserting job J_{k+1} in I'_t is a feasible transition for S_k.

2. Now we prove that (III) and (IV) hold after inserting the J_{k+1} using the inductive hypothesis. Suppose they hold after inserting k jobs, i.e.

$$W_k^{j*} \leq W_k^{j} \leq W_k^{j*}(1+\frac{\epsilon}{2m})^{\frac{k}{m}}, \quad 1 \leq j \leq K+1 \tag{4.3.18}$$

$$V_k^{j*} \leq V_k^{j} \leq V_k^{j*}(1+\frac{\epsilon}{2m})^{\frac{k}{m}}, \quad 1 \leq j \leq K+1 \tag{4.3.19}$$

For the job J_{k+1}, suppose it is inserted in I_t as a tardy job in the optimal sequence. Then from the above proof, it can also be inserted as a tardy job in interval I_t from the new problem. According to the optimal DP: $W_{k+1}^{t*} = W_k^{t*} + w_{k+1}$ and $W_{k+1}^{j*} = W_k^{j*}, j = 1, ..., K+1$ *and* $j \neq t$. Let $W'^{j}_{k+1}, j = 1, ..., K+1$ be the weights for $\mathcal{P}'$ after inserting J_{k+1} but before the rounding of Step 3. Then, $W'^{t}_{k+1} = W_k^t + w_{k+1}$ and $W'^{j}_{k+1} = W_k^j, j = 1, ..., K+1, j \neq t$. According to the rounding up of W'_j, we have

$$W_{k+1}^{t} \geq W'^{t}_{k+1} = W_k^t + w_{k+1} \overset{(4.3.18)}{\geq} W_k^{t*} + w_{k+1} = W_{k+1}^{t*}$$

$$W_{k+1}^{j} \geq W'^{j}_{k+1} = W_k^j \overset{(4.3.18)}{\geq} W_k^{j*} = W_{k+1}^{j*}, \quad j = 1, ...K\ \&\ j \neq t$$

$$\begin{aligned} W_{k+1}^{t} &\leq W'^{t}_{k+1}(1+\frac{\epsilon}{2m})^{\frac{1}{m}} \\ &= (W_k^t + w_{k+1})(1+\frac{\epsilon}{2m})^{\frac{1}{m}} \end{aligned}$$

$$\begin{aligned}
&\overset{(4.3.18)}{\leq} (W_k^{t*}(1+\frac{\epsilon}{2m})^{\frac{k}{m}} + w_{k+1})(1+\frac{\epsilon}{2m})^{\frac{1}{m}} \\
&\leq (W_k^{t*}(1+\frac{\epsilon}{2m})^{\frac{k}{m}} + w_{k+1}(1+\frac{\epsilon}{2m})^{\frac{k}{m}})(1+\frac{\epsilon}{2m})^{\frac{1}{m}} \\
&= (W_k^{t*} + w_{k+1})(1+\frac{\epsilon}{2m})^{\frac{k+1}{m}} \\
&= W_{k+1}^{t*}(1+\frac{\epsilon}{2m})^{\frac{k+1}{m}} \\
W_{k+1}^{j} &\leq W'^{j}_{k+1}(1+\frac{\epsilon}{2m})^{\frac{1}{m}} \\
&= W_k^{j}(1+\frac{\epsilon}{2m})^{\frac{1}{m}} \\
&\overset{4.3.18}{\leq} W_k^{j*}(1+\frac{\epsilon}{2m})^{\frac{k}{m}}(1+\frac{\epsilon}{2m})^{\frac{1}{m}} \\
&= W_{k+1}^{j*}(1+\frac{\epsilon}{2m})^{\frac{k+1}{m}}, \quad j = 1, ...K \ \& \ j \neq t
\end{aligned}$$

And this implies:

$$W_{k+1}^{j*} \leq W_{k+1}^{j} \leq W_{k+1}^{j*}(1+\frac{\epsilon}{2m})^{\frac{k+1}{m}}, \quad j = 1, ..., K+1$$

So (III) is proved when J_{k+1} is inserted tardy. When J_{k+1} is inserted early, all W_{k+1}s remain the same with W_k. We can easily prove (III) holds. Thus, (III) is proved. Similarly, we can prove (IV).

3. It remains to prove (4.3.4). According to step 1, after inserting the first job, the possible difference between Z_k and Z_k^* does not exceed δ due to the rounding of the objective function. Then (4.3.4) holds for $k = 1$. According to the inductive hypotheses, we have

$$Z_k \leq Z_k^* + \frac{k\epsilon}{2m} Z_k^* + k\delta \tag{4.3.20}$$

We distinguish between the cases of J_{k+1} being inserted tardy or early. Assume that it is true up to the placement of job $J_k, 1 \leq k \leq m-1$. If J_{k+1} is inserted tardy, and by using (4.3.4) and (III) from the inductive hypothesis, we get

$$Z_{k+1} \leq Z_k + (W_k^j + w_{k+1})p_{k+1} + \delta$$

$$\begin{aligned}
&\le Z_k^* + \frac{k\epsilon}{2m} Z_k^* + k\delta + (W_k^{j*}(1+\frac{\epsilon}{2m})^{\frac{k}{m}} + w_{k+1})p_{k+1} + \delta \\
&\le Z_k^* + (W_k^{j*} + w_{k+1})p_{k+1} + \frac{k\epsilon}{2m} Z_k^* + \frac{\epsilon}{2m} W_k^{j*} p_{k+1} + (k+1)\delta \\
&\le Z_{k+1}^* + \frac{k\epsilon}{2m} Z_{k+1}^* + \frac{\epsilon}{2m} Z_{k+1}^* + (k+1)\delta
\end{aligned}$$

where the first inequality takes into account the increase of Z_{k+1} by at most δ due to its rounding in Step 2, and the last inequality is due to the placement of J_{k+1} as tardy in sequence $\{S_k^*\}$ as well.

The case of J_{s+1} being inserted early is treated in exactly the same way.

The proof is done. □

In what follows we concentrate on the last elements S_m^*, S_m of the two sequences in the statement of Lemma 4.3.1. Equation (4.3.4) for $k := m$ implies

$$\begin{aligned}
Z_m &\le Z_m^* + \frac{\epsilon}{2} Z_m^* + m\frac{\epsilon Z_{lb}}{4m} \\
&\le (1+\frac{3\epsilon}{4}) Z_m^*
\end{aligned} \tag{4.3.21}$$

Note that Z_m in (4.3.21) is the objective value in $\mathcal{P}'$ *without* the guessed straddlers. We now study the effect on this value after inserting the straddlers. Let LP^*, LP be the two versions of (LP) we get using the data of S_m^* and S_m respectively. Let x^*, R^* and x, R be the solutions and objective values for LP^* and LP respectively (which we know that exist, since Conditions 2 and 2' are satisfied from Lemma 4.3.1). Then, after inserting the straddlers, the final schedule objective values are $Z' = Z_m + R$ for $\mathcal{P}'$ and $Z^* = Z_m^* + R^*$ for $\mathcal{P}$. Let Z^{extra} be the increase of the objective value when we go from $\mathcal{P}'$ back to $\mathcal{P}$ by moving the due dates d' back to the original d.[1] Let $Z = Z' + Z^{extra} = Z_m + R + Z^{extra}$ be the objective value of the schedule we output.

[1]We move only the due dates, not the jobs in the schedule, even if we can do the latter to our benefit.

Lemma 4.3.2 $R \leq (1+\frac{\epsilon}{2m})R^*$.

Proof. First note that the proof of Lemma 4.3.1 also proves (inductively) equation (4.3.12). This implies that x^* is also feasible for LP. Therefore

$$\begin{aligned} R &\leq \sum_{j=1}^{K} (V_m^j x_j^* + (w_{s_j} + W_m^{(j+1)})(p_{s_j} - x_j^*)) \\ &\overset{\text{(III)(IV)}}{\leq} \sum_{j=1}^{K} ((1+\frac{\epsilon}{2m})V_m^{j*} x_j^* + (w_{s_j} + (1+\frac{\epsilon}{2m})W_m^{(j+1)*})(p_{s_j} - x_j^*)) \\ &\leq (1+\frac{\epsilon}{2m})R^* \end{aligned}$$

The proof is done. □

We know that

$$d'_j - d_j \leq d'_K - d_K = (K-1)n\hat{\delta}, \quad j = 2, ..., K.$$

So the maximum possible extra increase of the objective going from $\mathcal{P}'$ to $\mathcal{P}$ is:

$$Z^{extra} \leq W(K-1)n\hat{\delta} \leq nw_{max}(K-1)n\hat{\delta} \leq K\frac{\epsilon Z_{lb}}{4m} \leq \frac{\epsilon}{4}Z^* \tag{4.3.22}$$

Then (4.3.21), Lemma 4.3.2, (4.3.22) imply

$$\begin{aligned} Z &= Z_m + R + Z^{extra} \\ &\leq (1+\frac{3\epsilon}{4})Z_m^* + (1+\frac{\epsilon}{2m})R^* + \frac{\epsilon}{4}Z^* \\ &\leq (1+\varepsilon)Z^* \end{aligned} \tag{4.3.23}$$

and our algorithm is an approximation scheme with the approximation ratio $1+\epsilon$.

4.3.3 Complexity of the Rounding Algorithm

We have proved that the algorithm outputs an approximate solution with approximation ratio $1+\epsilon$. Then in this section we will show the complexity of the rounding algorithm is polynomial in the problem size.

Lemma 4.3.3 *The algorithm runs in polynomial time.*

Proof. First we calculate the maximum number of states produced by Algorithm FPTAS. We have at most $O(n)$ distinct values for k, $O(\frac{n}{\varepsilon})$ distinct values for Z_k,

$$\frac{m \log W}{\log(1+\frac{\varepsilon}{2m})} = O(\frac{n^2}{\varepsilon}\log W)$$

distinct values for W_k, V_k. Also, there are at most

$$\frac{\frac{Z^{ub}}{w_{\pi(N)}}}{\hat{\delta}} \leq \frac{\frac{Z^{ub}}{w_{min}}}{\hat{\delta}} = \frac{n^2 w_{max} Z^{ub}}{\delta w_{min}} = O(\frac{n^3}{\epsilon}\frac{w_{max}}{w_{min}})$$

distinct intervals where y_k, e_k fall in. Therefore, for the same $Z_k, W_k^1, V_k^1, ..., W_k^{K+1}, V_k^{K+1}$, we have $O((\frac{n^3}{\varepsilon}\frac{w_{max}}{w_{min}})^{2K})$ states, and the total number of states is upper-bounded by

$$O\left(\frac{n^2}{\varepsilon}\left(\frac{n^5}{\varepsilon^2}\log W \frac{w_{max}}{w_{min}}\right)^{2K}\right)$$

If T_{LP} is the time needed to solve (LP), and considering that there are $O(n^K)$ possible guesses for the K straddlers, and $O(\log n + \log w_{max} + \log p_{max})$ guesses for Z^{ub}, the running time for our algorithm is upper-bounded by

$$O\left(\left(\frac{n^2}{\varepsilon}\left(\frac{n^5 w_{max}\log W}{\epsilon^2 w_{min}}\right)^{2K} + T_{LP}\right) n^K(\log n + \log w_{max} + \log p_{max})\right)$$

which is polynomial on n and $1/\varepsilon$ due to Assumption 1.1.1. □

From Section 4.3.2 and Section 4.3.3, we have shown the FPTAS for the minimum TWET problem with constant number of due dates and polynomially related weights.

4.4 A Simpler Rounding Scheme

4.4.1 A New Dynamic Programming

In Section 4.2 above, we define the state in the DP as

$$(k, Z_k; V_k^1; y_k^2, W_k^2, e_k^2, V_k^2; \dots; y_k^{K+1}, W_k^{K+1}, e_k^{K+1}, V_k^{K+1})$$

Notice that we don't keep the earliness and tardiness in the first interval, i.e, e_k^1 and y_k^1 are not kept in the state. We use the summation of the earliness and tardiness in all the other intervals to calculate the free space in the first interval according to equation 4.2.1. This makes us consider the first interval as a special one when doing the rounding scheme. Such a special consideration has made the FPTAS quite complicated. Therefore, to simplify the rounding scheme and the corresponding FPTAS, we can keep the earliness and tardiness information in the first interval and the new state in stage k of the dynamic programming becomes

$$(k, Z_k; y_k^1, W_k^1, e_k^1, V_k^1; y_k^2, W_k^2, e_k^2, V_k^2; \ldots; y_k^{K+1}, W_k^{K+1}, e_k^{K+1}, V_k^{K+1})$$

where k, Z_k, y_k, W_k, e_k and V_k have the same meaning with the DP in Section 4.2. Notice that $y_k^1 = W_k^1 = e_k^{K+1} = V_k^{K+1} = 0$ since no job is tardy in the first interval and no job is early in the last interval.

The $m = n - K$ non-straddling jobs are sorted in the I-WSPT order and will be processed the same way as the DP in Section 4.2. Similarly, we define $L_k^{(j-1)j}$ to be the free space in interval I_j, $1 \leq j \leq K+1$ in stage $0 \leq k \leq m$. Then, we have

$$L_k^{(j-1)j} = \begin{cases} d_j - d_{j-1} - (y_k^j + e_k^j), & 1 \leq j \leq K \\ \infty, & j = K+1 \end{cases} \tag{4.4.1}$$

Notice that since we keep the earliness and tardiness in the first interval, we can calculate the free space in the intervals using the same formula for $1 \leq j \leq K$, while in equation 4.2.1 we need a special formula to calculate L_k^{01}.

By replacing the free space $L_k^{(j-1)j}$ defined in equation 4.2.1 in Section 4.2 with the new one given in equation 4.4.1, the quantities u_k^i and the two conditions needed to check the feasibility of inserting the job are defined the same way as those in Section 4.2.

With the two feasibility conditions, we follow the same transition from state $(k, Z_k; \ldots)$ to state $(k+1, Z_{k+1}; \ldots)$ as that in Section 4.2.

Similarly, if at some point $Z_{k+1} > Z^{ub}$ is true for all possible insertions of J_{k+1} then we reject Z^{ub}, we replace it with a new $Z^{ub} := 2Z^{ub}$, and start the algorithm from scratch.

Placement of the straddlers

Similarly to the insertion of straddlers in Section 4.2, we need to insert the K straddlers after calculating the states of stage m through a linear problem. The new linear problem is the same with (LP) except replacing $L_m^{j(j+1)}$ with the one in equation (4.4.1). It is quite straightforward that the new linear problem also has an optimal solution using a similar proof to Lemma 4.2.1.

4.4.2 The New FPTAS

The DP in Section 4.4.1 computes an optimal schedule in pseudo-polynomial time. In this section we will use a new simpler rounding scheme to produce an FPTAS for the new DP in Section 4.4.1.

First, let $\varepsilon > 0$ be the approximation parameter of the FPTAS. Define $Z_{lb} := Z^{ub}/2$ and $\delta = \frac{\varepsilon Z_{lb}}{4m}$. The interval $[0, Z^{ub}/w_{min}]$ is divided into $x = \lceil \frac{Z^{ub}}{\hat{\delta} w_{min}} \rceil$ subintervals $\{H_i\}_{i=1}^{x}$ of length $\hat{\delta} = \frac{\delta}{n^2 w_{max}}$ (note that the length of the last subinterval may be smaller than $\hat{\delta}$). These actions are the same with those in Section 4.3.

Different from Section 4.3, we change the original problem $\mathcal{P}$ into a new problem $\mathcal{P}'$ with exactly the same set of jobs, but with new due dates $d'_j, j = 1, ..., K$ defined as follows:

$$d'_j = d_j + 2jn\hat{\delta}, \; j = 1, \ldots, K \tag{4.4.2}$$

Then we have new intervals $I'_j, j = 1, ..., K+1$. Obviously, we also extend the first interval, while the due date $d'_1 = d_1$ in equation 4.3.3.

From now on, we use Condition 1 and Condition 2 to stand for the feasibility checking conditions described in Section 4.4.1 with the new L given in equation 4.4.1. Similarly, we can define the free space L' in each interval, the values u', and the conditions for feasible transitions exactly in the same way as in Section 4.4.1 using the new due dates. We will also refer to the latter as Conditions 1' and 2'. We will also need to compare states S produced by the FPTAS to states S^* produced by the exact DP; we will use the asterisk $(^*)$ to denote quantities that belong to S^*, in order to distinguish them from quantities that belong to S.

The new simpler DP algorithm with rounding scheme is the following:

Simpler Algorithm FPTAS

Step 1'. Order all m jobs in I-WSPT order. The initial state is $(0, 0, ..., 0)$.

Step 2'. Each state in stage $0 \leq k \leq m$

$$(k, Z_k; y_k^1, W_k^1, e_k^1, V_k^1; y_k^2, W_k^2, e_k^2, V_k^2; \ldots; y_k^{K+1}, W_k^{K+1}, e_k^{K+1}, V_k^{K+1})$$

will generate at most $K+1$ new states (feasible transitions), one for each placement of job J_{k+1} to interval I_j, $j = 1, \ldots, K+1$. In these new states, we round up Z_{k+1}, W_{k+1}^j and V_{k+1}^j the same way as in Step 2 in Figure 4.5.

Step 3'. For each state produced in Step 2', the values y_{k+1}^1, e_{k+1}^1, y_{k+1}^2, e_{k+1}^2, $\ldots$, y_{k+1}^{K+1}, e_{k+1}^{K+1} fall in subintervals H_i; call the tuple of these subintervals the *signature* of the state. We group together the states that have *the same values* of $Z_{k+1}, W_{k+1}^1, V_{k+1}^1, \ldots, W_{k+1}^{K+1}, V_{k+1}^{K+1}$ and *the same signature*. From each group, we *randomly* pick a state as a representative and keep it; all the transitions going to the group states we discard become transitions to the state we keep.

Figure 4.6: The Simpler FPTAS Algorithm

Notice that the big difference between this new DP with a rounding scheme and the one in Section 4.3.1 lies in Step 3'. Different from Step 3 in

Figure 4.5 in which we keep only the state that maximizes $\sum_{j=2}^{K+1}(y_{k+1}^{j}+e_{k+1}^{j})$ for the states with the same signature, in the new Step 3' we keep a random one among the states with the same signature. Since we also extend the first interval according to equation 4.4.2, we don't need to keep the state which maximizes $\sum_{j=2}^{K+1}(y_{k+1}^{j}+e_{k+1}^{j})$ to make more space in the first interval and just pick a random one. Similarly, we have the following new lemma to guarantee the approximation ratio which is similar to Lemma 4.3.1 in Section 4.3.2.

Lemma 4.4.1 *Let $\{S_k^*\}_{k=0}^{m}$ be the sequence of states produced by the optimal placement of jobs in the new exact DP of Section 4.4.1. Then the same sequence of placements is also feasible for Algorithm FPTAS, and the sequence of states $\{S_k\}_{k=0}^{m}$ produced satisfies*

$$Z_k \le Z_k^* + \frac{k\epsilon}{2m} Z_k^* + k\delta,\ \forall k. \tag{4.4.3}$$

Proof. Using induction.

For $0 \le k \le m$, in addition to (4.3.4), the following four inductive hypothesis also hold for step k:

$$(y_k^j + e_k^j) - (y_k^{j*} + e_k^{j*}) \le 2k\hat{\delta}, \quad 1 \le j \le K+1 \tag{i}$$

$$W_k^{j*} \le W_k^j \le W_k^{j*}(1+\frac{\epsilon}{2m})^{\frac{k}{m}}, \quad 1 \le j \le K+1 \tag{ii}$$

$$V_k^{j*} \le V_k^j \le V_k^{j*}(1+\frac{\epsilon}{2m})^{\frac{k}{m}}, \quad 1 \le j \le K+1 \tag{iii}$$

A feasible placement of job J_{k+1} from S_k^* to S_{k+1}^* is also feasible from S_k to S_{k+1}. (iv)

For convinience, below we use L, u instead of L', u'.

For the base case ($k = 0$), first notice that (4.4.3),(i), (ii),(iii),(iv) hold trivially.

We assume that the inductive hypothesis is true up to the placement of job $J_k,\ 0 \le k \le m-1$.

1. First we will prove (i): For inductive step $k+1$, suppose the optimal DP sequence places J_{k+1} in interval I_t. Conditions 1 & 2 imply

$$L_k^{(t-1)t*} \geq p_{k+1} \tag{4.4.4}$$

$$L_{k+1}^{j(j+1)*} \geq u_{k+1}^{j*},\ j = 1, ..., K-1. \tag{4.4.5}$$

Let ${y'}_{k+1}^{j}$ and ${e'}_{k+1}^{j}$ be the tardiness and earliness after inserting J_{k+1} in I'_t but before the rounding according to Step 3' of the algorithm. After Step 2', we know that

$$\begin{aligned} {y'}_{k+1}^{t} + {e'}_{k+1}^{t} =&\ y_k^t + e_k^t + p_{k+1} \\ {y'}_{k+1}^{j} + {e'}_{k+1}^{j} =&\ y_k^j + e_k^j,\ j = 1, ..., K+1\ \&\ j \neq t \end{aligned}$$

and similarly for the optimal DP. Therefore

$$({y'}_{k+1}^{j} + {e'}_{k+1}^{j}) - (y_{k+1}^{j*} + e_{k+1}^{j*}) = (y_k^j + e_k^j) - (y_k^{j*} + e_k^{j*}) \overset{\text{(i)}}{\leq} 2k\hat{\delta} \tag{4.4.6}$$

Since state $S_{k+1}$with y_{k+1}^j, e_{k+1}^j and state S'_{k+1} with ${y'}_{k+1}^{j}, {e'}_{k+1}^{j}$ have the same signature $(H^1, H^2, ...)$ and each H^i interval has length $\hat{\delta}$, we have

$$|y_{k+1}^j - {y'}_{k+1}^{j}| \leq \hat{\delta},\ \ |e_{k+1}^j - {e'}_{k+1}^{j}| \leq \hat{\delta}$$

and, therefore,

$$\begin{aligned} (y_{k+1}^j + e_{k+1}^j) - (y_{k+1}^{j*} + e_{k+1}^{j*}) &\leq ({y'}_{k+1}^{j} + {e'}_{k+1}^{j}) - (y_{k+1}^{j*} + e_{k+1}^{j*}) + 2\hat{\delta} \\ &\overset{(4.4.6)}{\leq}\ 2(k+1)\hat{\delta} \end{aligned} \tag{4.4.7}$$

So the inequalities (i) holds after inserting J_{k+1}. Thus it hold $\forall j \in [1, K+1],\ k \in [0, m]$.

2. Combined with inequality (4.4.1), it is easy to see that

$$\begin{aligned} L_k^{(j-1)j} - L_k^{(j-1)j*} =&\ (d'_j - d'_{j-1} - y_k^j - e_k^j) - (d_j - d_{j-1} - y_k^{j*} - e_k^{j*}) \\ =&\ 2n\hat{\delta} - [(y_k^j + e_k^j) - (y_k^{j*} + e_k^{j*})] \geq\ 0 \qquad j = 1, 2...K+1 \end{aligned}$$

which implies that $L_k^{(j-1)j} \geq L_k^{(j-1)j*} \overset{(4.4.4)}{\geq} p_{k+1}$, and Condition 1' holds. and similarly

$$L_{k+1}^{(j-1)j} \geq L_{k+1}^{(j-1)j*} \tag{4.4.8}$$

Here we assert that $u_{k+1}^{j*} \geq u_{k+1}^{j}, \ \ j = 0, 1, 2...K$.

The assertion is trivial when $j = 0$. Suppose it is true for some j, where $j \geq 0$, then

$$\begin{aligned} u_{k+1}^{j+1} &= \max\{0, p_{s_{j+1}} - L_{k+1}^{j(j+1)} + u_{k+1}^{j}\} \\ u_{k+1}^{(j+1)*} &= \max\{0, p_{s_{j+1}} - L_{k+1}^{j(j+1)*} + u_{k+1}^{j*}\} \end{aligned}$$

since $L_{k+1}^{j(j+1)*} \leq L_{k+1}^{j(j+1)}$, and $u_{k+1}^{j*} \geq u_{k+1}^{j}$, thus $u_{k+1}^{(j+1)*} \geq u_{k+1}^{(j+1)}$ holds. So,

$$u_{k+1}^{j*} \geq u_{k+1}^{j}, \forall j = 0, 1, ..., K \tag{4.4.9}$$

(4.4.5),(4.4.8),(4.4.9) show that

$$L_{k+1}^{j(j+1)} \geq L_{k+1}^{j(j+1)*} \geq u_{k+1}^{j*} \geq u_{k+1}^{j}, for\ all\ j = 1, 2..., K-1$$

which means that Condition 2' also holds. Hence inserting job J_{k+1} in I'_t is a feasible placement for S_k, and (iv) is proven.

3. Using the inductive hypothesis, it is easy to see that (ii) and (iii) hold after Step 2' using the same way to prove equation (III) and (IV) in Lemma 4.3.1.

4. It remains to prove (4.4.3). We distinguish between the cases of J_{k+1} being inserted tardy or early.

   If J_{k+1} is inserted tardy, and by using (4.4.3) and (ii) from the inductive hypothesis, we get

   $$Z_{k+1} \leq Z_k + (W_k^j + w_{k+1})p_{k+1} + \delta$$

$$\begin{aligned}
&\leq Z_k^* + \frac{k\epsilon}{2m} Z_k^* + k\delta + (W_k^{j*}(1+\frac{\epsilon}{2m})^{\frac{k}{m}} + w_{k+1})p_{k+1} + \delta \\
&\leq Z_k^* + (W_k^{j*} + w_{k+1})p_{k+1} + \frac{k\epsilon}{2m} Z_k^* + \frac{\epsilon}{2m} W_k^{j*} p_{k+1} + (k+1)\delta \\
&\leq Z_{k+1}^* + \frac{k\epsilon}{2m} Z_{k+1}^* + \frac{\epsilon}{2m} Z_{k+1}^* + (k+1)\delta
\end{aligned}$$

where the first inequality takes into account the increase of Z_{k+1} by at most δ due to its rounding in Step 2', and the last inequality is due to the placement of J_{k+1} as tardy in sequence $\{S_k^*\}$ as well.

The case of J_{s+1} being inserted early is treated in exactly the same way. Then (4.4.3) is proven.

The proof is done. □

Concentrating on the last elements S_m^*, S_m of the two sequences in the statement of Lemma 4.4.1 above, and following similar steps to insert the straddlers back, we can prove that the final output $Z \leq (1+\epsilon)Z^*$ where Z^* is the optimal solution for the original problem. The new DP does not increase the complexity of the algorithm. According to Lemma 4.3.3, the new DP also runs in polynomial time.

Therefore, the new DP gives an FPTAS for the TWET problem with a constant number of distinct due dates and polynomially related weights.

4.4.3 General Straddlers

Remember that for the FPTAS in Figure 4.5 and the new one in Figure 4.6, we have guessed K straddlers for the K distinct due dates. Here there is the assumption that each straddler just straddles one due date. We show that our algorithm can be easily extended to the case where some straddler straddles many due dates. If there is one straddler which straddles more than one due date, denoted as S_{many}, we guess the number of consecutive due dates it straddles; then we guess the due dates positions for S_{many}; at last we guess a job as S_{many}. We also guess the other straddlers for other due dates. Since S_{many} straddles due dates from d_i to d_j, $1 \leq i < j \leq K$, no other jobs can be inserted between times d_i and d_j. We can add such a condition for the DP to reduce the number of states. We insert the non-straddling jobs using the DP. Finally, we re-insert these straddlers back. If there are many straddlers which straddle many due dates, we apply exhaustive enumeration to guess all of them and the due dates they straddle. For each of them, we add the condition that no jobs can be inserted between the two due dates it straddles. When K is a constant, the number of such combinations of straddlers is still polynomial. Then following the same rounding scheme, we can also get an FPTAS.

Chapter 5

Conclusions and Open Questions

5.1 Conclusion

In this thesis, for the minimum total weighted earliness and tardiness problem with a constant number of distinct due dates and polynomially related weights, we have presented a pseudo-polynomial time dynamic programming algorithm to solve it and then changed it into an FPTAS through a rounding scheme.

We notice that the straddlers play special roles in the schedule. They separate the early and tardy jobs and all the non-straddling jobs become either early or tardy. Through exhaustive enumeration, we guess all possible combinations of the straddlers from all the n job. For each guessing, we use dynamic programming to get a partial schedule of all the remaining non-straddling jobs. Before the insertion of the non-straddling jobs, we order them in the I-WSPT order because of the structural properties of the optimal schedule of the TWET problem.

For each non-straddling job, we try to insert it into all the intervals. It is a tardy job when it's inserted in the intervals after its due date, otherwise it will be an early job. We reduce the number of states by keeping only the feasible states. Thus, we come up with two feasibility checking conditions. One is to check if there's enough space in some interval to insert a job in.

The other one is to check if there is enough space to insert all the straddlers after inserting the job. With these two conditions, we start to insert the jobs one by one while keeping all the states feasible. After the insertion of all the non-straddling jobs, we insert the straddlers back to get the final schedule.

To make the running time of the algorithm polynomial, we need to apply a rounding scheme to reduce the number of states. Compared with the TWT problem with constant number of distinct due dates in [21], no preemption is allowed in our TWET problem. Thus, here we increase the due dates a little bit to get a new problem. In the new problem, the free space in each interval becomes larger and thus the preemption can be avoided. We apply a rounding scheme for the DP for the new problem with new due dates, and insert the straddlers back. Eventually, we move the due dates back to get the solution for the original TWET problem. In such a way, an FPTAS has been developed to solve our problem. We have shown two different ways to increase the due dates and they influence the rounding scheme. Thus, we give two FPTAS with two different rounding schemes according to the two increases in due dates. One is in Section 4.3 and a simpler version is given in Section 4.4.

5.2 Open Questions

Although, in reality, many practical problems belong to the TWET problem with a constant number of due dates and polynomially related weights we are focusing on, we still need to consider the more general cases, like the TWET with arbitrary number of distinct due dates, or with arbitrary weights, or with non-symmetric weights, or even their combinations. Such problems are still open. We know that for the TWET problem with a constant number of distinct due dates and arbitrary weights, we can use the DP in Section 4.2 to get an optimal solution in pseudo-polynomial time. How to design an FPTAS is open for future work.

Bibliography

[1] T.S. Abdul-Razaq and C.N. Potts. Dynamic programming state-space relaxation for single-machine scheduling. In *Journal of the Operational Research Society*, pp. 141–152, 1988.

[2] T.S. Abdul-Razaq, C.N. Potts and L.N. Van Wassenhove. A survey of algorithms for the single machine total weighted tardiness scheduling problem. In *Discrete Applied Mathematics*, Vol. 26(2), pp. 235–253, 1990.

[3] M.S. Akturk and M.B. Yildirim. A new lower bounding scheme for the total weighted tardiness problem. In *Computers and Operations Research*, Vol. 25(4), pp. 265–278, 1998.

[4] E.M. Arkin and R.O. Roundy. Weighted-tardiness scheduling on parallel machines with proportional weights. In *Computers and Operations Research*, Vol. 39(1), pp. 64–81, 1991.

[5] P. Babu, L. Peridy and E. Pinson. A branch and bound algorithm to minimize total weighted tardiness on a single processor. In *Annals of Operations Research*, Vol. 129(1-4), pp. 33–46, 2004.

[6] K.R. Baker. Introduction to sequencing and scheduling. Wiley, NY, 1974.

[7] K.R. Baker and G.D. Scudder. Sequencing with earliness and tardiness penalties: a review. In *Operations Research*, Vol. 38(1), pp. 22–36, 1990.

[8] P. Brucker. Scheduling algorithms. Springer, 2007.

[9] T.C.E. Cheng, C.T. Ng, J.J. Yuan and Z. Liu. Single machine scheduling to minimize total weighted tardiness. In *European Journal of Operational Research*, Vol. 165(2), pp. 423–443, 2005.

[10] R.K. Congram, C.N. Potts and S.L. Van De Velde. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. In *INFORMS Journal on Computing*, Vol. 14(1), pp. 52–67, 2002.

[11] S.E. Elmaghraby. The one machine sequencing problem with delay costs. In *Journal of Industrial Engineering*, Vol. 19, pp. 105–108, 1968.

[12] H. Emmons. One-machine sequencing to minimize certain functions of job tardiness. In *Operations Research*, Vol. 17(4), pp. 701–715, 1969.

[13] M.R. Garey, R.E. Tarjan, and G.T. Wilfong. One-processor scheduling with symmetric earliness and tardiness penalties. In *Mathematics of Operations Research*, Vol. 13, pp. 330–348, 1988.

[14] V. Gordon, J. Proth and C. Chu. A survey of the state-of-the-art of common due date assignment and scheduling research. In *European Journal of Operational Research*, Vol. 139(1), pp. 1–25, 2002.

[15] R. Hassin and M. Shani. Machine scheduling with earliness, tardiness and non-execution penalties. In *Computers and Operations Research* vol. 32, pp. 683–705, 2005.

[16] N.G. Hall and M.E. Posner. Earliness-tardiness scheduling problems I: Weighted deviation of completion times about a common due-date. In *Operations Research*, Vol. 39(5), pp. 836–846, 1991.

[17] N.G. Hall, W. Kubiak, and S.P. Sethi. Earliness–tardiness scheduling problems, II: deviation of completion times about a restrictive common due date. In *Operations Research*, Vol. 39(5), pp. 847–856, 1991.

[18] J.A. Hoogeveen. Multicriteria scheduling. In *European Journal of Operational Research*, Vol. 167, pp. 592–623, 2005.

[19] P.A. Huegler and F.J. Vasko. A performance comparison of heuristics for the total weighted tardiness problem. In *Computers and Industrial Engineering*, Vol. 32(4), pp. 753–767, 1997.

[20] I. Kacem. Fully polynomial time approximation scheme for the total weighted tardiness minimization with a common due date. In *Discrete Applied Mathematics*, Vol. 158, pp. 1035–1040, 2010.

[21] G. Karakostas, S. Kolliopoulos and J. Wang. An FPTAS for the minimum total weighted tardiness problem with a fixed number of distinct due dates. In *Computing and Combinatorics*, pp. 238–248, 2009.

[22] H. Kellerer and V.A. Strusevich. A fully polynomial approximation scheme for the single machine weighted total tardiness problem with a common due date. In *Theoretical Computer Science*, Vol. 369(1), pp. 230–238, 2006.

[23] H. Kellerer and V.A. Strusevich. Minimizing total weighted earliness-tardiness on a single machine around a small common due date: an FPTAS using quadratic knapsack. In *International Journal of Foundations of Computer Science*, pp. 357–383, 2009.

[24] H. Kellerer and V.A. Strusevich. Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications. In *Algorithmica*, Vol. 57(4), pp. 769–795, 2010.

[25] S.G. Kolliopoulos and G. Steiner. Approximation algorithms for minimizing the total weighted tardiness on a single machine. In *Theoretical Computer Science*, Vol. 355(3), pp. 261–273, 2006.

[26] M.Y. Kovalyov and W. Kubiak. A fully polynomial approximation scheme for the weighted earliness-tardiness problem. In *Operations Research*, Vol. 47, pp. 757–761, 1999.

[27] E.L. Lawler. A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. In *Annals of Discrete Mathematics*, Vol. 1, pp. 331-342, 1977.

[28] E.L. Lawler. A fully polynomial approximation scheme for the total tardiness problem. In *Operations Research Letters*, Vol. 1(6), pp. 207-208, 1982.

[29] J.K. Lenstra, A.H.G. Rinnooy Kan and P. Bruker. Complexity of machine scheduling problems. In *Econometric Institute of the Erasmus University*, 1977.

[30] J.Y.T. Leung. Handbook of scheduling: algorithms, models, and performance analysis. Chapman & Hall/CRC, Vol. 1, 2004.

[31] J.C. Picard and M. Queyranne. The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. In *Operations Research*, Vol. 26(1), pp. 86–110, 1978.

[32] M.L. Pinedo. Scheduling: theory, algorithms, and systems. Springer, 2012.

[33] J.C. Picard and M. Queyranne. A branch and bound algorithm for the total weighted tardiness problem. In *Operations Research*, Vol. 33(2), pp. 363–377, 1985.

[34] R.M.V. Rachamadugu. Technical Note - Note on the Weighted Tardiness Problem. In *Operations Research*, Vol. 35(3), pp. 450-452, 1987.

[35] L. Schrage and K.R. Baker. Dynamic programming solution of sequencing problems with precedence constraints. In *Operations Research*, Vol. 26(3), pp. 444-449, 1978.

[36] J. Shwimer. On the N-job one-machine, sequence-independent scheduling problem with tardiness penalties: A branch-bound solution. In *Management Science*, Vol. 18(6), pp. B–301, 1972.

[37] M. Müller-Hannemann and A. Sonnikow. Non-approximability of just-in-time scheduling. In *Journal of Scheduling*, Vol. 12(5), pp. 555–562, 2009.

[38] P.S. Sundararaghavan, and M.U. Ahmed. Minimizing the sum of absolute lateness in single-machine and multimachine scheduling. In *Naval Research Logistics Quarterly*, Vol. 31(2), pp. 325–333, 1984.

[39] S. Verma and D. Dessouky. Single-machine scheduling of unit-time jobs with earliness and tardiness penalties. In *Mathematics of Operations Research*, Vol. 23, pp. 930–943, 1998.

[40] V.V. Vazirani. Approximation algorithms. Springer, 2004.