JING WANG

AN FPTAS FOR THE SINGLE MACHINE MINIMUM TOTAL WEIGHTED TARDINESS PROBLEM WITH A FIXED NUMBER OF DISTINCT DUE DATES

BY
JING WANG, B.Sc., M.Sc.

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfilment of the Requirements
for the degree
Master of Applied Science

McMaster University

MASTER OF APPLIED SCIENCE (2009) McMaster University, Hamilton, Ontario

TITLE: An FPTAS for the Single Machine Minimum Total Weighted Tardiness Problem With a Fixed Number of Distinct Due Dates

AUTHOR: Jing Wang, M.Sc.(Wilfrid Laurier University)

SUPERVISOR: professor George Karakostas

NUMBER OF PAGES: x, 68

Abstract

This thesis provides a Fully Polynomial Time Approximation Scheme (FPTAS) for the minimum total weighted tardiness (TWT) problem with a constant number of distinct due dates.

Given a sequence of jobs on a single machine, each with a weight, processing time, and a due date, the tardiness of a job is the amount of time that its completion time goes beyond its due date. The TWT problem is to find a schedule of the given jobs such that the total weighted tardiness is minimized. This problem is NP-hard even when the number of distinct due dates is fixed. In this thesis, we present a dynamic programming algorithm for the TWT problem with a constant number of distinct due dates first and then adopt a rounding scheme to obtain an FPTAS.

Three major points that we make in this algorithm are: we observe a series of structural properties of optimal schedules so that we shrink the state space of the DP; we make use of preemption (i.e. allowing the processing of a job to be interrupted and restarted later) for the design of the DP; the rounding scheme that we adopt guarantees that a factor $1+\epsilon$ of the optimal solution is generated and the algorithm runs within a polynomial time of the problem size.

Acknowledgements

I gratefully acknowledge professor George Karakostas for his advice, supervision, and crucial support, which made him a backbone of this research and so to this thesis. His involvement with his creativity and patience has greatly triggered my enthusiasm in doing research. I am sure that I will benefit from his advice in my future academic life.

I wish to express my warm and sincere thanks to professor Antoine Deza, professor Ned Nedialkov and professor Sanzheng Qiao. I believe one of the main gains of this master programm was taking their courses and gaining their advices. I would also thank for professor Franya Franek and professor Tamas Terlaky for their suggestions and encouragement. I would like to express my deep appreciation to professor Tim Davidson and professor Michael Soltys who give me a lot of valuable suggestions on improving my thesis.

Many thanks go in particular to my friends Xiaoxi Ma, Jiaping Zhu, Feng Xie, Bingzhou Zheng, Shefali Kulkarni, Bahareh Mansouri and so on. They are helpful and supportive friends. I have had a great time with them.

At last, I would like to thank my families: , my parents in China, my husband and my mentor professor Michael J. Best. Without their supports, I would not be able to successfully complete this master's program. Particularly, I would like to give my utmost gratitude to professor Michael J. Best. He is always there to give me supports and encouragement whenever I need. His father-like guide helps me and Lujing in both work and life aspects. We are lucky to have him in our lives.

Contents

List of Figures

Chapter 1

Introduction

With the objective of minimizing (maximizing) one or more than one performance measures, Scheduling is the allocation of scarce resources to various activities [Leung2004]. Scheduling problems are very complicated since they are generally operated in multi-machine environments. Practically, these multi-machine problems are usually decomposed into subproblems which are operated in single machines. If the single machine scheduling problem is solved, the result can be used to understand and provide the basis for the solution of more complicated problems. Also the results can help us model both single machine and multi-machine environments problems [Pinedo1995]. In this thesis, we are particularly interested in the problem of minimizing the total weighted tardiness (TWT) on a single machine.

1.1 The TWT problem and the motivation

First of all, let's review the definition of the TWT problem. In a single machine environment, we are given n jobs. Each job is indexed as $1, 2, \ldots, n$. Assume that all the n jobs are ready at time 0 for processing on a machine without interruption. For each job J_j where

$j = 1, 2, \ldots, n$, there is a weight for the delay penalty, denoted as $w_j > 0$. The weight measures how important the job is. There is a processing time p_j which is assumed to be a positive integer since it can be normalized anyway. There also is a due date d_j which stands for the point in time by which the processing of job J_j is due to be completed.

With the assumption that only one job can be processed at a time on the single machine, every arrangement of the order of the n jobs identifies one scheduling. In a given scheduling (in other words, a processing order), each job has a time at which processing that job is finished. As a convention, this time is defined as the completion-time C_j of this job. For each job, the amount of time by which the completion time of this job exceeds its due date is called the lateness L_j. Mathematically it is formulated as $L_j := C_j - d_j$. Obviously when a job is completed before its due date, then the lateness is negative, while if a job is processed after it is due date, then the lateness is positive. Further, the tardiness of a job is defined as the lateness of this job if it fails to meet its due date, or zero otherwise. It is formulated as $T_j := \max\{0, L_j\}$. Tardiness is a measure of comparing the actual completion-time with the desired completion-time (i.e., its due date) [Conway1967]. When a job's tardiness is positive, this job is called a tardy job. Each tardy job J_j is associated with a weighted tardiness $w_j T_j$ where w_j is the delay penalty per unit of tardiness. In this thesis, we study the minimization of total weighted tardiness (TWT), i.e., $\min\{\sum_{j=1}^{n} w_j T_j\}$, when the number of distinct due dates is constant.

The TWT problem attracts the attention of both the industry and academic researchers. This problem on a single machine is fundamentally important within a network of operations in industrial fields, for example, the semiconductor wafer fabrication industry, heat treatment operations in the aircraft industry, the large steel casting industry and so on [Kanet2007]. If those operations can be sequenced effectively, the overall facility performance will be improved significantly [Mathirajan2006]. The TWT problem has received much attention not only because it is meaningful in practice, but also because it can be

used to test the newly designed scheduling techniques or algorithms for efficiency or accuracy purposes [Kellerer2006]. Some scheduling problems are either special cases of the TWT problem or closely related to it. For example, when all the weights are assumed to be 1, then minimizing TWT becomes minimizing the total tardiness; if we know some jobs are tardy then minimizing the total weighted tardiness of these jobs is the same as minimizing the total weighted completion time. All these problems are known to be NP-hard [Du1990] [Rinnooy Kan1976] [Lawler1977] [Yuan1992]. If the TWT problem can be solved, then a series of related problems can be solved too.

In literature, there are only a few results involving arbitrary due dates and arbitrary weights that have been reported because of the hardness of the problem. So far, only some special cases of the TWT problem have been intensively researched [Cheng2005], for instance, when all the jobs have a common due date [Kahlbacher1993] [Kellerer2006] [Lawler1977]; on the equal-slack due date model, i.e., $d_j = p_j + q$, where q is a given constant [Cheng2005] [Oguz1994] [Qi1998]; on the processing-plus-wait model [Cheng2005], when all jobs are assumed to have the same weight; when all the jobs have the same processing time; when the maximum weight is bounded polynomially, and so on. In this thesis, we focus on a model where a constant number of distinct due dates are assigned to all the jobs.

1.2 Previous Work

Although, for arbitrary positive weights, the minimum TWT problem is known to be strongly NP-hard, various aspects of solving the minimum TWT problem and its variants have attracted considerable attention from researchers. Therefore, we present some of their results first. There are numerous results, but we can only cite a limited number of them here. With the purpose of making the introduction clearer, we classify these works in the

following four categories.

1.2.1 Branch-and-bound

The branch-and-bound method was firstly applied to solve the TWT problem by Elmaghraby [Elmaghraby1968]. The paper showed the advantage of branch-and-bound method over the dynamic programming method on efficiency. Because of the inapplicability of the algorithm given by Elmaghraby on the computer, Shwimer [Shwimer1972] proposed a branch-and-bound algorithm for the TWT problem which is applicable on a computer. Since then, the application of branch and bound on the TWT problem has been widely explored. Picard et al. [Picard1978] observed that the time-dependent traveling salesman problem can be considered as the TWT problem. They used the method of finding shortest paths in a network with sub-gradient optimization and the method of branch and bound enumeration to solve the TWT problem and gave computational results.

The sub-gradient optimization technique is well known for calculating the lower bound of the optimal solution. Potts and Van Wassenhove [Potts1985] proposed a even faster one which is called the multiplier adjustment method to compute the lower bound of the optimal solution. By combining checking for some dynamic programming dominance properties, Potts et. al. proposed a branch and bound algorithm for the TWT problem. Compared to all previous algorithms that can solve the TWT problem with at most 20 jobs, the computational results of this algorithm show a big advantage. However, because of the difficulty of the TWT problem itself, this algorithm still can not work efficiently or runs into severe trouble when the problem size goes above 50 [Congram2002].

Dominance rules were first developed by Emmons [Emmons1969]. They were established for the minimizing total tardiness problem, i.e., $\min\{\sum_{j=1}^{n} T_j\}$, to restrict the size of the search and find optimal solutions. By using those dominance rules, Rachamadugu

[Rachamadugu1987] showed a proposition that states a local dominance property among adjacent jobs in an optimal schedule. This kind of precedence relationship has been already tested by Potts et al. [Potts1985] in their branch-and-bound algorithm. With this property, the first job in an optimal schedule can be determined without solving the TWT problem. Most importantly this proposition can identify the weighted shortest processing time (WSPT) relationship among adjacent tardy jobs. The WSPT rule is important throughout the whole thesis, so we emphasize it here.

In [Potts1985], Potts et al. tried to formulate the TWT problem as a Lagrangian problem and then decomposed it into two subproblems. The algorithm that they proposed the use of the Lagrangian dual solution value as a lower bound on the TWT problem. However, Hoogeveen et al. [Hoogeveen1995] found a stronger lower bound by reformulating the Lagrangian problem. More specifically, they introduced a vector of slack variables to the Lagrangian problem such that a modified slack variable problem with equality constraints was formulated, and they tested that an improved lower bound on the optimal solution was obtained by the new algorithm.

However, the lower bound on the optimal solution was still not practical to use because of the extensive computation the algorithm needs. Therefore, Akturk and Yildirim [Akturk1998] developed a new lower bounding scheme which introduced a new dominance rule that can guarantee a sufficient condition for a local optimum for the TWT problem. This new dominance rule is a generalization of the rules in [Emmons1969] and [Rinnooy Kan1976]. It considers the time dependent orderings between any pair of jobs and hence finds tighter lower and upper bounds. The use of this new dominance rule can reduce the number of alternatives for finding the optimal solution. These computational experiments show that this dominance rule can greatly improve the lower bound of the optimal solution.

Babu et al. [Babu2004] performed 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 designed a Lagrangian heuristic which provides an upper bound on the calculated solution. Using the above lower and upper bounds, they gave a branch-and-bound algorithm. In their algorithm, in order to reduce the job domain, they presented a collection of dominance and elimination rules which helps them to find a trade-off between a tighter lower bound and the time needed for the enumeration process. The computational experiments show that this algorithm works very efficiently on the instances of 40 to 50 jobs.

The short coming of branch-and-bound is that it needs a great amount of computational time.

1.2.2 Dynamic programming method

A very early dynamic programming algorithm for sequencing problems with precedence constraints was given by Schrage [Schrage1978]. Unlike Emmons who proposed dominance rules, Schrage et al. considered precedence constraints which are either from practical considerations or from the characteristics of an optimal sequence, with the same purpose, that is, to reduce the number of sequences to be considered. Based on the enumeration of all feasible subsets of tasks, they presented a method to assign an easily computed label to each feasible subset. These labels actually provide physical addresses to store the subset information. As a result, a compact computer implementation of dynamic programming algorithms can be applied to many scheduling problems, including the TWT problem.

Arkin and Roundy [Arkin1991] considered a special model of the TWT problem where the weight of each job is proportional to its processing time. They proposed a pseudo-polynomial time algorithm for solving this special case and gave a bound on a ratio which is used to measure the deviation from optimality. On the other side, Yano and Kim [Yano1991] developed an optimal procedure that uses dominance properties to reduce

the number of sequences and constructs good initial sequences. Computational experiments show that the heuristic solutions are greatly improved when a pairwise interchange procedure is used. Szwarc and Liu [Szwar1993] claimed that their two stage decomposition process can solve this problem completely or reduce the problem into smaller subproblems. Working with cluster decomposition method, their algorithm can solve 150 out of 320 tested problems completely.

Huegler et al. [Huegler1997] proposed a dynamic programming based heuristic. Using the same method as in [Potts1991] to generate random problems, they performed a great amount of testing to compare their new heuristic to other classic heuristics. In these tests, the new method requires less computation time than others, especially for large size problems, although the solution quality is not as good as the ones obtained by other heuristics for some test problems.

A big improvement in greatly reducing the complexity has been made recently by Congram [Congram2002]. He proposed the idea of dynasearch, a new neighborhood search technique, which overcomes the limited searching ability of traditional descent algorithms. Instead of using lexicographic search, dynasearch uses dynamic programming in local search. It allows multi-moves in each iteration and explores exponential size neighborhoods in polynomial time, and hence it dramatically decreases the computation time. Also since it performs a sufficiently large number of random moves from local minimum, dynasearch stands a much better chance to find a better solution than other techniques.

Although this method can generate exact solutions, it is restricted by computer storage requirements, especially when the number of jobs is more than 50.

1.2.3 Other heuristics

Although the first attempts at solving the TWT problem used branch-and-bound, dynamic programming methods, or their combinations based on implicit enumeration techniques, a large number of other heuristics have also been researched. In literature, the main heuristics techniques are: construction techniques and interchange techniques. More specifically, the construction method builds sequences by fixing a job at each step, while the interchange method, like the name suggests, starts with an existing sequence and continues to interchange parts of the sequence. Typically, heuristic methods for the TWT problem are tested on randomly generated problems [Huegler1997].

Potts [Potts1991] presented a collection of heuristics, from simple to complex ones. Through a large set of test problems, they compared the proposed heuristics to interchange and simulated annealing methods. Their tests show that the straightforward interchange method performs better than other heuristics and simulated annealing methods for the TWT problem.

Nevertheless, after testing many search techniques, such as tabu search, simulated annealing, descent, threshold search and genetic algorithms using binary coding schemes, Crauwel et al. [Crauwels1998] found that permutation methods were more likely to produce an optimal solution than binary-based methods, but binary-based methods consistently generated good quality solutions whereas permutation-based methods did not. Their computational results also showed that tabu search gave the best overall results in terms of the solution quality and computation time [Sen2003].

There is another heuristic technique that has been widely studied, called the meta-heuristic method. It is formally defined as an iterative generation process which guides and modifies the operations of subordinate heuristics by combining intelligently different concepts for exploring and exploiting the search space to efficiently produce high quality

solutions. Learning strategies are used to structure information in order to find efficiently near-optimal solutions [Osman1996]. Meta-heuristic algorithms are approximate and usually non-deterministic.

Madureira [Madureira1999] used the idea of "exchanging jobs which are not apart more than a given number of positions" and performed a great deal of testing on evaluating several kinds of heuristic methods. The computational results illustrate the robustness and flexibility of the meta-heuristic procedures. Based on the Ant Colony Optimization (ACO) meta-heuristic, Besten et al. [Besten2000] presented an algorithm which is claimed to generate good solutions for the TWT problem. A powerful local search algorithm, candidate lists and a heterogeneous ant colony are considered as the three key elements for the development of this effective ACO algorithm.

Borgulya [Borgulya2002] described a three-stage cluster-based evolutionary algorithm for the TWT problem. It is a combination of construction heuristics, local search and meta-heuristics. In the first stage, the local optimal solutions are estimated by grouping. In the second stage, the accuracy of results is improved by a local search procedure. Then, in the last stage, the estimations are further refined. This algorithm finds the best-known solution within an acceptable time limit comparing with other heuristic methods.

Liu et al. [Liu2003] proposed a genetic algorithm for the TWT problem which uses the natural permutation representation of a chromosome, the combination of heuristic dispatching rules and random methods to create the initial population, position-based crossover and order-based mutation operators, and the selection of the best members of the population during generations. All these techniques make the genetic algorithm to be performed better than decent methods in obtaining good solutions.

1.2.4 Approximation algorithms

Heuristics may generate reasonably close or even optimal solutions but this is not guaranteed. They provide a basic idea for approximation algorithms . With certain degree of loss in optimality, approximation algorithms can compute the near optimal solution very efficiently. The difference between approximate algorithm and heuristic is that approximate algorithm produces solutions with a worst case performance guarantee in both computational time and solution quality. Nowadays, not only branch-and-bound, dynamic programming and heuristic algorithms are intensively researched, but also approximation algorithms are receiving increasing attention for solving combinatorial optimization problems.

In particular, researchers have applied approximation algorithms to solve the TWT problem. With the assumption that the weights of jobs are agreeable, i.e, $p_i < p_j$ implies $w_i > w_j$, Lawler [Lawler1977] developed a "pseudo-polynomial" time dynamic programming algorithm for the TWT problem. When the weights of all the jobs are the same, Lawler [Lawler1982] gave a fully polynomial time approximation scheme (FPTAS) which is a modification of his pseudo-polynomial dynamic programming algorithm. When the due dates are assumed to be linear functions of their processing times and the tardiness weights are assumed to be proportional to their processing times, Cheng et al. [Cheng2005] provided a polynomial-time $n-1$-approximation algorithm. When the due dates are assumed to have equal slacks, i.e. $d_i = p_i + q$ where q is a constant, they constructed an FPTAS.

For arbitrary weights and arbitrary due dates, however, frustratingly little is known on the approximability of TWT. When all the jobs are assumed to have the same due date, Fathi and Nuttle [Fathi1990] discussed the performance of four polynomially bounded heuristics. The fourth heuristics gave an approximation algorithm with an approximation ratio of 2. Then Kellerer and Strusevich [Kellerer2006] give an FPTAS for this case.

For the case of a fixed number of due dates, Kolliopoulos and Steiner [Kolliopoulos2006] developed a pseudo-polynomial dynamic programming algorithm whose complexity depends on the total processing time. Assuming that the maximum job weight is bounded by a polynomial of n, where n is the number of given jobs, they treated the problem as a bin packing problem, allowed preemption for the early jobs, modified the above algorithm and adapted Lawler's [Lawler1982] rounding scheme so that they generated an FPTAS. Moreover, when the due dates are assumed to be concentrated around small values, and the maximum processing time is bounded by a polynomial of n, they developed a quasi-polynomial algorithm.

1.3 Our Work and Thesis Outline

In this thesis, we develop an FPTAS for the TWT problem with a fixed number of distinct due dates. Most of the results of this thesis appear in [Karakostas2009]. Briefly speaking, we design a pseudo-polynomial algorithm first and then apply a rounding scheme to obtain the desired approximation scheme. This work is greatly inspired by [Kellerer2006], but introduces a number of new ideas for this problem.

To understand our algorithm, a key idea is to "think backwards". Suppose that we know which sequence is optimal and observe the properties that it has. Assume that every straddler (the job starting before or on a due date and completing after a due date) is only straddling one due date. It is easy to see that the straddlers are either early (finishes before its own due date) or tardy (finishes after its own due date). We note that early straddlers are always surrounded by early jobs and the order of early straddlers and early jobs with the same due date can be arbitrary. On the other hand, tardy straddlers are always followed by tardy jobs. This indicates that we should distinguish tardy straddlers from all other jobs. In reality, we do not know which sequence is optimal. How do we know that which due

dates correspond to tardy straddlers in the optimal sequence? Which jobs should be the tardy straddlers? A very useful and accurate technique, although not efficient, is exhaustive enumeration (or guessing). So from now on, we say that we "have guessed" the tardy straddlers.

We continue our "think backwards" process on the optimal schedule. All the tardy straddlers divide the time horizon into several super-intervals. We notice that in each super-interval all the tardy jobs must be processed before all the early jobs since only the tardy ones contribute to the objective value. It has been proved that for these tardy jobs, the WSPT order minimizes the total weighted tardiness. So the tardy jobs in each super-interval must be in WSPT order. This compels us to sort the input jobs at the beginning in WSPT order in our algorithm. We do not sort the tardy jobs after inserting them because we do not want to keep record of which job goes where, since this will make the algorithm exponential anyway no matter how we round up variables later. Instead, we need only to record which jobs are early and which are tardy.

Next we consider the placement of the jobs including the tardy straddlers. Kolliopoulos and Steiner [Kolliopoulos2006] place the straddlers by guessing their positions at the beginning of their algorithm. But we follow Kellerer's FPTAS which inserts them after inserting the other jobs. But we do not know how to insert the rest of the jobs, how do we know where to place the straddlers? So we continue to "think backwards". In the optimal sequence, we remove tardy straddlers one-by-one starting from the last one. It is easy to see that the last tardy straddler is on the last due date. After removing it, there is a hole crossing the last due date. Push the whole job block in the last super-interval backward up to the last due date. We notice that all the jobs are still tardy, although each of them sees a decrement of its tardiness by the same amount. Then we move on to the next tardy straddler and do the same thing. Again, we find that no placement (early or tardy) of the other jobs is violated. We keep doing this until we remove the earliest tardy straddler.

Now we are left with a partial schedule which has only $n - M$ jobs. Note that the placement of these jobs are exactly the same as their placement in the optimal sequence. What has changed is only the tardiness of each tardy job. This partial schedule will be refereed to as an "abstract schedule". This partial schedule is an extension of the"stop due date problem" in [Kolliopoulos2006]. We follow Kellerer et al. [Kellerer2006] to construct a "stop due date"-like problem for the non-straddling jobs. But the situation we have now is much more complicated than the one in [Kellerer2006]. For example, if a job is tardy, it can be tardy in many super-intervals. We note that there are actually many nice properties in the optimal sequence. For example, in each super-interval, if there are tardy jobs, they must gather in the first interval, where interval is the time between two consecutive due dates. If a job is early, it only goes to one super-interval which contains the due date of this job.

Now suppose that we do not know the partial schedule which gives us an optimal schedule in the end. We have to try all possible placements in order to find it. It is easy to decide where a job should go, but we must also check whether there is room for it. In order to check this, we come up with sufficient feasibility conditions. If a job is to be inserted as tardy, then the insertion should not cause any previously inserted early job tardy. If a job is to be inserted early, then there must be enough room before its due date in certain super-interval. Since we are going to use a rounding scheme in the FPTAS, we have to make sure that those conditions are satisfied. When the straddlers are inserted back, they should not turn any early job into tardy and they should not push any tardy job beyond the first interval in any super-interval. Once all these conditions are satisfied, a job can be placed safely. More specifically, we place a tardy job after the last tardy one in a super-interval (the first one starting from the due date); we place an early job anywhere before its due date.

Note that the feasibility conditions must also hold for the FPTAS, where we use pre-

emption. Preemption is to allow the interruption of the processing of a job at any point in time and the placing of a different job on the machine instead. When a preempted job is afterwards put back on the machine, it only needs the machine for its remaining processing time. Preemption is needed because our dynamic programming algorithm uses enumeration and hence its complexity is exponential on the size of the input. To reduce the running time, we round up the total processing time of tardy jobs with common due date in each super-intervals. One consequence of this rounding is that some jobs which are tardy in optimal sequences can not be inserted in that super-interval anymore and some jobs already early in that super-interval become tardy. However, following [Kellerer2006], we notice that for each due date, the total processing time of tardy and early jobs with that due date is fixed, which means that when we round up the total processing time of the tardy jobs of a common due date, the total processing time of the early jobs with the same due date is actually rounded down by the same amount. Therefore, if we allow preemption for the tardy jobs, we can use the space saved from the early jobs. Since preemption is allowed, we revise the feasibility conditions so that we are still able to find a near optimal solution.

We emphasize that in order to make complexity polynomial in the problem size, we have to consider a reduced the number of values of the variables by rounding up. However, we can not reduce them arbitrarily. We have to confine our output within a neighborhood of the optimal value. Ultimately, it is the feasibility conditions that help us to achieve this goal. Applying this FPTAS to the TWT problem with one common due date, it is easy to see that the algorithm in [Kellerer2006] is a special case of our algorithm. We develop a fully polynomial time approximation scheme for the TWT problem with a constant number of distinct due dates. But the algorithm does not run in fully polynomial time necessarily for the TWT problem with an arbitrary number of distinct due dates. When the number of distinct due dates is arbitrary, then the complexity is exponential.

To summarize, the procedure of our FPTAS works as following: order the input n

jobs in weighted shortest processing time order, i.e. $\frac{p_1}{w_1} \leq \frac{p_2}{w_2} \leq \ldots \leq \frac{p_m}{w_m}$; guess K number of tardy straddlers (exhaustive enumeration of the due dates and jobs) where the tardy straddler is defined as the job starting before its due date but finishes after the same or a different due date; the guessed K due dates partition the time line into $K + 1$ super-intervals (defined as the space between two consecutive tardy straddlers); for each set of guessed tardy straddlers, we apply the following dynamic programming algorithm for the non-straddling jobs:

- For each job, guess it as either early or tardy by selecting the proper super-interval. If it is inserted as early, then it is inserted non-preemptively and explicitly. If it is inserted as tardy, then it can be inserted preemptively. But the job is left "floating" from its completion time backwards. To place the job successfully, we have to check certain feasibility conditions. If a placement can be done, we apply a rounding scheme (given in Chapter 4) to the new calculation of the state variables, it corresponds to a DP feasible transition from a state to another.

When all the non-straddling jobs are inserted, we insert the tardy straddlers into the correct positions preemptively, for similar reasons as for the tardy non-straddling jobs.

The remainder of this thesis is organized as follows. In Chapter 2, basic background of approximation algorithms and some related scheduling problems will be introduced. In Chapter 3, we introduce the key ideas of [Kellerer2006] which are crucial for our work. Chapter 4 is the main result of this thesis, the solution of the TWT problem with a constant number of due dates by an FPTAS. Then in the last chapter, we give a summary of the whole thesis and possible future work.

Chapter 2

Approximation Scheme and Some Scheduling Problems

Through the introduction in chapter 1, we saw that approximation algorithms can be used to solve NP-hard scheduling problems. Here if a algorithm finds a solution within the approximation guarantee, then we say that we "solve" the problem. In this chapter we are going to introduce some basic concepts of approximation algorithms. On the other hand, some basic scheduling problems are related to each other. If an optimal schedule for one problem can be found, it may help us to find an optimal schedule for another related problem. For this purpose, we introduce some elementary results of the optimality of some scheduling problems.

2.1 Overview of approximation algorithms

2.1.1 Basic concepts

Many natural optimization problems, including those arising in important application areas, are NP-hard. Therefore, under the widely believed conjecture that $P \neq NP$, their exact solution is prohibitively time consuming. If we do not require the exact value, and there is an efficient algorithm that can produce a provably good solution, then that may also be satisfying. This is the idea of approximation algorithms, that is, generate provably near-optimal solutions efficiently in terms of complexity. Since the 1970s, approximation algorithms have become more prominent as the tool for the generation of near optimal solutions for NP-hard optimization problems.

In the following paragraphs, we will quote the exact mathematical definitions of the main concepts in the area of approximation algorithms. Details can be found in [Vazirani2001]. An optimization problem is specified by a set $\mathcal{I}$ of inputs, a set $S(I)$ of feasible solutions for every input $I \in \mathcal{I}$, and an objective function f that specifies for every feasible solution s in $S(I)$ an objective value $f(s)$. 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 instance I by $|I|$, i.e., the number of bits used in writing down I in some fixed encoding. The TWT problem that we are dealing with is NP-hard, therefore it is unlikely to be able to find the exact optimal solution within time polynomial in $|I|$.

Polynomial time approximation schemes

Let **P** be a minimization problem. Let $\rho \geq 1$. An algorithm A is called a ρ-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 ratio of the approximation algorithm.

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

Pseudo-Polynomial time approximation schemes

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

Definition 2 Pseudo-Polynomial Time is polynomial time where integers in the input string are given in unary representation rather than in binary.

Fully polynomial time approximation schemes

Definition 3 A Fully Polynomial Time Approximation Scheme (FPTAS) for problem **P** is a PTAS whose time complexity is polynomial in the input size and $\frac{1}{\epsilon}$.

With respect to worst case approximation, an FPTAS is the strongest possible result that we can hope to derive for an NP-hard problem. The only difference between FPTAS and PTAS is the requirement on polynomial time in $\frac{1}{\epsilon}$.

In order to establish the approximation guarantee, the output of the approximation algorithm needs to be compared with the optimal objective value. However, we do not know the optimal objective value, or how to find the optimum in polynomial time. Therefore, the basic problem is to establish the approximation guarantee. The general strategy in the

area of approximation algorithms is to find a lower bound for the optimal solution of the minimization problem and relate the approximation algorithm output to the lower bound. In Chapters 3 and Chapter 4, we will see how such a lower bound can be chosen for the approximation algorithm for the TWT problem.

In our problem, the size of the problem is the number of inputs, which includes the number of the jobs n, each job's weight w and processing time p. To find an optimal schedule of the TWT problem, one needs to evaluate $n!$ sequences if enumeration is used. Dynamic programming is another option in this case. Even though the computational demands of dynamic programming grow at an exponential rate with the increase of the problem size, this method is typically more efficient than complete enumeration. It considers certain sequences indirectly while enumeration evaluates all the sequences explicitly. Dynamic programming can find the optimal, but it may still need exponential time. Therefore we have to think of an approximation algorithm which uses dynamic programming, but limits its enumeration (i.e. number of states) to polynomial.

2.2 Basic results in single machine scheduling

The optimal schedule of some scheduling problems has a special structure. For instance, the optimal sequence for the Minimum Mean Flow-time problem follows the Shortest Processing Time (SPT) rule; the optimal sequence for the Minimum Total Weighted Completion time problem is in the Weighted Shortest Processing Time (WSPT) rule; and the optimal sequence for the Maximum Tardiness problem follows the Earliest Due Date (EDD) rule. These will be described below. In these cases, knowledge of an optimum pairwise job ordering allows the optimal schedule to be constructed with a simple sorting mechanism. These results can be used directly to find the solution of some practical scheduling problems in certain situations. Also, they can be used to understand the optimal schedule for

other more complicated scheduling problems, such as the Minimum TWT problem.

In this section, we mainly introduce two rules: EDD and WSPT. We only introduce these two because an optimal schedule for the TWT problem is constructed from a combination of EDD and WSPT. For purpose of completion, we include the proofs together with the lemmas. For more details, one is referred to [Baker1974] and [Pinedo1995]. The maximum job lateness problem is defined as

$$L_{max} = \max_{1 \leq j \leq n} \{L_j\} \tag{2.1}$$

where L_j is the amount of time by which the completion time of job J_j exceeds its due date. The maximum job tardiness is defined as

$$T_{max} = \max_{1 \leq j \leq n} \{T_j\} \tag{2.2}$$

where T_j is the lateness of job J_j if it fails to meet its due date, or zero otherwise. The minimum total weighted completion time is defined as

$$\min_{\textit{schedule of n jobs}} \{\sum_{j=1}^{n} w_j C_j\} \tag{2.3}$$

where C_j is the completion time of job J_j.

Earliest due date rule

Definition 4 The Earliest Due Date (EDD) rule is to order the jobs in earliest due date order, that is, $d_1 \leq d_2 \leq \ldots \leq d_n$.

We have the following:

Lemma 1 The maximum job lateness (L_{max}) and the maximum job tardiness (T_{max}) are minimized by EDD sequencing.

Proof. (from [Baker1974]) We employ the method of adjacent pairwise interchange. Consider a sequence S that is not the EDD sequence. That is, somewhere in S there must exist a pair of adjacent jobs, i and j, with j following i, such that due date $d_i > d_j$. Now construct a new sequence, S', in which jobs i and j are interchanged and all other jobs complete at the same time as in S. Then

$$L_i(S) = t_B + p_i - d_i \quad , \quad L_j(S') = t_B$$
$$L_j(S) = t_B + p_i + p_j - d_j \quad , \quad L_i(S') = t_B + p_j + p_i - d_i$$

from which it follows that $L_j(S) > L_i(S)$ and $L_j(S) > L_j(S')$ where t_B is the point of time where the set of jobs preceding job i and job j is complete. Hence

$$L_j(S) > \max\{L_i(S'), L_j(S')\}$$

Let $L + \max\{L_k | k \in A \quad or \quad k \in B\}$ and notice that L is the same under both S and S'. Then

$$L_{max}(S) = \max\{L, L_i(S), L_j(S)\} \geq \max\{L, L_i(S'), L_j(S')\} = L_{max}(S')$$

In other words the interchange of jobs i and j does not increase the value of L_{max}, and may actually reduce it. A similar argument will establish that EDD minimizes T_{max}, beginning

with the inequality

$$T_{max}(S) = \max\{0, L_{max}(S)\} \geq \max\{0, L_{max}(S')\} = T_{max}(S').$$

□

Weighted shortest processing time rule

Definition 5 The Weighted Shortest Processing Time (WSPT) rule is to order the jobs so that the first job has the shortest processing time per unit of weight, the second processed the next shortest processing time per unit of weight, and so on, that is, $\frac{p_1}{w_1} \leq \frac{p_2}{w_2} \leq \ldots \leq \frac{p_n}{w_n}$, where p_i and w_i are the processing time and weight of job i, $\forall 1 \leq i \leq n$.

We have the following lemma:

Lemma 2 The WSPT rule is optimal for the single machine minimum total weighted completion time problem, as defined in (2.3).

Proof. (from [Baker1974]) By contradiction. Suppose a schedule S, that is not WSPT, is optimal. In this schedule there must be at least two adjacent jobs, say job j followed by job k, such that $\frac{w_j}{p_j} < \frac{w_k}{p_k}$. Assume job j starts its processing at time t. Perform a so-called adjacent pairwise interchange on job j and k. Call the new schedule S'. While under the original schedule S job j starts its processing at time t and is followed by job k, under the new schedule S' job k starts its processing at time t and is followed by job j. All other jobs remain in their original position. The total weighted completion time of the jobs processed before jobs j and k is not affected by the interchange. Neither is the total weighted completion time of the jobs processed after jobs j and k. Thus the difference in the values of the objectives under schedules S and S' is due only to jobs j and k. Under S the total weighted completion time of jobs j and k is $(t + p_j)w_j + (t + p_j + p_k)w_k$ while

under S' it is $(t+p_k)w_k + (t+p_k+p_j)w_j$. It is easily verified that if $\frac{w_j}{p_j} < \frac{w_k}{p_k}$ the sum of the two weighted completion times under S' is strictly less than under S. This contradicts the optimality of S and completes the proof of the theorem. □

Chapter 3

FPTAS for the Single Machine TWT Problem with Common Due Date

Kellerer and Strusevich [Kellerer2006] proposed an FPTAS for the single machine TWT problem with a common due date. We will use some of their ideas to develop an FPTAS for the TWT problem with a constant number of distinct due dates. In this chapter, we present briefly their work.

3.1 Construct a stop due date problem

With the one due date assumption, for n jobs, there must be one job straddling the due date, the so called *straddler*, that starts before or on the due date and completes after it. Kellerer et al. [Kellerer2006] guess the straddler at the beginning of their algorithm. Instead of inserting the straddler by guessing the starting time like [Kolliopoulos2006] do, they insert first the $n - 1$ non-straddling jobs. In this section, we present "a stop due date problem" which is constructed for the non-straddling jobs. Note that, whenever there is a guessed straddler, there is a stop due date problem associated with it.

It is easy to see that any non-straddling job finishing before the due date is early and any non-straddling job starting after the due date is tardy. Therefore, the due date separates all non-straddling jobs into two categories: early and tardy jobs. The stop due date problem is constructed by processing jobs as follows:

1. all the non-straddling jobs are ordered by the WSPT rule and are placed in the time line one by one;
2. the first early job starts from time zero and the first tardy job starts from the due date;
3. all early jobs are processed as a block without intermediate idle time and completed by the due date;
4. all tardy jobs are processed as a block without intermediate idle time.

The goal of this problem is to find the minimum total weighted tardiness of the remaining $m := n - 1$ jobs for each chosen straddler.

3.2 DP for the stop due date problem

For the stop due date problem, the total weighted tardiness is calculated by $Z_m = \sum_{j=1}^{m} w_j T_j$. It is easy to see that only the tardy jobs are counted for the calculation of the total weighted tardiness. It has been proved in [Kolliopoulos2006] that the WSPT rule minimizes the total weighted tardiness of a set of tardy jobs. To minimize the total weighted tardiness for the stop due date problem, it is important to decide which jobs are scheduled as tardy. Therefore, a Boolean decision variable x_j is defined for each job J_j, where $x_j = 1$ if job j is tardy and $x_j = 0$ otherwise. Then the total weighted tardiness for a feasible schedule for the stop due date problem is given by $Z_m = \sum_{j=1}^{m} w_j(\sum_{i=1}^{j} p_i x_i)x_j$. Here "feasible" means that the total processing time of the 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; $y_k := \sum_{j=1}^{k} p_j x_j$ is the total processing time of the tardy jobs among the first k jobs; $W_j := \sum_{j=1}^{k} w_j x_j$ is the total weight of these jobs; $Z_k = \sum_{j=1}^{k} w_j (\sum_{i=1}^{j} p_i x_i) x_j$ is the current value of the objective function.

After the decisions have been made for the first k jobs, the state of the dynamic programming is defined as (k, Z_k, y_k, W_k). We say that all the states with the same k are "from stage k". Then the transition from a state (k, Z_k, y_k, W_k) in a stage to any state $(k+1, Z_{k+1}, y_{k+1}, W_{k+1})$ in the next stage is defined as follows:

1. if the next job J_{k+1} in the WSPT order is decided to be early, and feasibility condition $A_k - y_k \leq d$ where $A_k = \sum_{j=1}^{k} p_j$, then $y_{k+1} = y_k$, $W_{k+1} = W_k$, and $Z_{k+1} = Z_k$;
2. if it is decided to be tardy, then $y_{k+1} = y_k + p_{k+1}$, $W_{k+1} = W_k + w_{k+1}$, and $Z_{k+1} = Z_k + w_k y_{k+1}$

The initial state is defined as $(0, 0, 0, 0)$.

The DP above generates a collection of states after considering all $n-1$ jobs. Between the initial state any one of the states in the last stage, if there is a connected path connects them, then it must be a feasible path, since every state in this path is calculated only when a job a feasibly inserted. Here a connected path means that, for all the $n-1$ states (one from each stage) on this path, there is a transition for any two consecutive states. The corresponding schedule is called a feasible schedule. This DP generates all the feasible paths for each stop due date problem.

3.3 Inserting the straddler back

Recall that before applying DP to the stop due date problem, we have guessed a straddler. Now it is time for it to be scheduled back into each of the feasible schedules. More specifically, the straddler is processed starting from the completion time of the last early job in

the time line. To make the insertion successfully, the tardy block has to be pushed forward. It is easy to calculate the amount of processing time by which the tardy block is pushed. It is $x = A_m - y_m + p_s - d$, where p_s is the processing time of the straddler. The tardiness of all the tardy jobs are actually increased by a mount of x. Since the tardiness of the straddler is also x, we compute the total weighted tardiness for the TWT problem with common due date as $Z = Z_m + w_s x + W_m x$. For each stop due date problem, we can find a schedule which gives the minimum Z. After considering all the stop due date problems (one for each possible straddler), we find an optimal schedule for the TWT problem with common due date.

Note that for each stop due date problem, all the feasible schedules should be considered when the straddler is inserted back. It is wrong to only consider the feasible schedule which gives the minimum total weighted tardiness for the stop due date problem. This can be easily seen from the formulation $Z = Z_m + w_s x + W_m x$. A minimum Z_m does not mean a minimum Z because of the $(w_s + W_m)x$ part.

3.4 Rounding scheme

In the worst case, the dynamic programming above generates $\Theta(2^n)$ states which is way too inefficient when the problem size is relatively large. The goal of the rounding scheme is to reduce the number of states in each stage so that eventually there are only a polynomial number of states generated. In order to reduce the number of states, it is necessary to reduce the number of values that each variable takes. On the other side, we do not want to lose too much accuracy. Therefore, we show that the complexity is a polynomial of the problem size, and at the same time, the solution is bounded by a factor of $1 + \epsilon$ of the optimal solution. Here $1 + \epsilon$ is the approximation ratio. To do this, [Kellerer2006] defines the following rounding scheme:

1. use the objective value of a heuristic schedule for the original problem as an upper bound, denoted by Z^{ub}. This value is of approximation ratio 2;

2. for any $\epsilon > 0$, define $Z_{lb} = 1/2Z^{ub}$ and $\delta = \frac{\epsilon Z_{lb}}{4m}$;

3. for each $Z_k \leq Z^{ub}$, round it up to the next multiple of δ;

4. for each W_k, round it up to the nearest power of $\tau := (1 + 1/\epsilon)^{1/m}$;

5. sort all the N distinct weights in decreasing order, then we have $\frac{Z^{ub}}{w_{\pi(1)}} < \frac{Z^{ub}}{w_{\pi(2)}} < \ldots < \frac{Z^{ub}}{w_{\pi(N)}}$ where $\pi(i)$ is the decreasing order of the distinct weights. Split interval $[0, \frac{Z^{ub}}{w_{\pi(n)}}]$ at these values. Then we have $I_1 = [0, \frac{Z^{ub}}{w_{\pi(1)}}]$, $I_2 = [\frac{Z^{ub}}{w_{\pi(1)}}, \frac{Z^{ub}}{w_{\pi(2)}}]$, $\ldots$, $I_N = [\frac{Z^{ub}}{w_{\pi(N-1)}}, \frac{Z^{ub}}{w_{\pi(N)}}]$. Further divide each interval I_i where $1 \leq i \leq N$ into sub-intervals of length $\frac{\delta}{w_{\pi(j)}}$. Denote the sub-interval by I_j^r where r is its index;

6. for each combination of states (having the same value of Z_k and W_k) and a subinterval I_j^r, determine the smallest and largest values of y_k. We only save two states (k, Z_k, y_k^{min}, W_k) and (k, Z_k, y_k^{max}, W_k).

In this algorithm, the most important part is the feasibility conditions, i.e., $A_k - y_k \leq d$ for all $1 \leq k \leq m$. We know that in any feasible schedule for the stop due date problem, all tardy jobs are scheduled in the interval $[d, \infty]$ and all early jobs are scheduled in the interval $[0, d]$. There is always enough space in interval $[d, \infty]$ for placing tardy jobs. But for the early jobs, we have to check if the empty space left from placing the previous early jobs in interval $[0, d]$ is enough for placing another early job. The empty space left from placing early job among the first k jobs is $d - (A_k - y_k)$. Here [Kellerer2006] uses that the total processing time of the first k jobs is the sum of the total processing time of the early jobs among the first k jobs plus the total processing time of the tardy jobs among the first k jobs.

In the FPTAS, y_k is always rounded up, therefore the corresponding total processing time of the early jobs among the first k jobs e_k is rounded down. Therefore, every time y_k is rounded up, the empty space left to the k-th job $(d - e_k)$ is increased. Therefore, if a job is early in an optimal schedule, the FPTAS can also place it as early. Since there is always plenty space for tardy jobs, if a job is tardy in an optimal schedule, the FPTAS can place it as tardy. This eventually guarantees that we would not lose more than a factor $1 + \epsilon$ of the optimal, where $1 + \epsilon$ is the approximation ratio.

Chapter 4

FPTAS for Single Machine TWT Problem with a Fixed Number of Distinct Due Dates

In this chapter, we define a stop due dates problem and further define an abstract problem by analyzing the structural properties of an optimal schedule for the TWT problem with a constant number of distinct due dates. We give a dynamic programming algorithm for the abstract problem and then convert it into an FPTAS by adopting the rounding scheme in [Kellerer2006].

4.1 A stop due dates problem

In this section, we consider the TWT problem with a constant number K of distinct due dates. Sort them in increasing order, i.e., $d_1 < d_2 < \ldots < d_K$. Jobs with the same due date can be grouped into one class and hence there are totally K job classes. This chain of due dates partitions the time horizon into $K + 1$ intervals. We define artificial due

dates $d_0 = 0$ and $d_{K+1} = \infty$. Then the intervals can be represented as $I_i = [d_{i-1}, d_i)$ where $1 \leq i \leq K+1$. As defined in the common due date case, a job processed before or on a due date but completed after it (or a different due date) is called a straddler. When a straddler finishes after a different due date, we say this straddler spans more than one due dates. For ease of exposition, we will assume that there is an optimal schedule with distinct straddlers for every due date, that is, there are K distinct straddlers $S_1, S_2, \ldots, S_K$ corresponding to due dates $d_1, d_2, \ldots, d_K$ respectively. Later in this chapter, we will explain how to modify the algorithms to deal with the case of some straddlers crossing more than one due date.

It has been proved in [Kolliopoulos2006] that, in any optimal schedule, the non-straddling tardy jobs scheduled consecutively in any interval I_i ($i > 1$) must appear in WSPT order. Therefore, we order the given n jobs in WSPT order at the beginning and later on we place jobs in this order one by one. As in [Kellerer2006], we guess K straddlers. "Guess" means exhaustive enumeration. For the remaining $m = n - K$ jobs, still in WSPT order, we define *a stop due dates problem*. In this problem, in each interval I_i the first tardy job (if there is one) starts at the first due date. The block containing early jobs starts after the tardy block (if there is one) and completes before or at the second due date of this interval. Both early and tardy blocks are consecutive.

The early block in any interval is scheduled after the tardy block due to the argument in [Kolliopoulos2006], that is, in an optimal schedule, all the early jobs in each interval are scheduled after all the tardy jobs (if there are any). Note that if tardy and early blocks already exist in an interval and another job is to be placed as tardy in this interval, we need to move the early block forward so that this tardy job can be inserted right after the tardy block, as the figure 4.1 shows. If a job is to be placed as early in an interval, we simply place it right after the last early job, as the figure 4.2 shows.

We define the following quantities which are very important throughout this chapter:

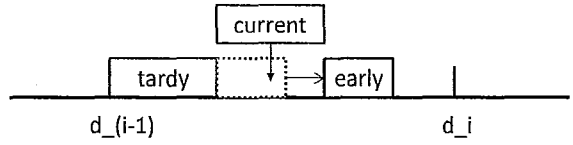

Figure 4.1: Inserting a job as tardy.

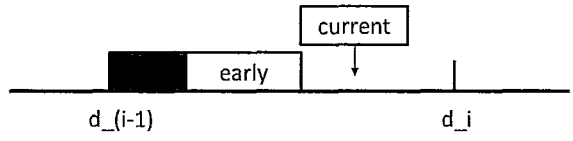

Figure 4.2: Inserting a job as early .

1. $y_k^{(i-1)t}, 1 \leq t < i \leq K+1, 1 \leq k \leq m$: the total processing time of the tardy jobs among the first k jobs that belong to class C_t and are processed in I_i.

2. $W_k^{(i-1)t}, 1 \leq t < i \leq K+1, 1 \leq k \leq m$: the total weight of the jobs in the previous item.

3. $A_k^t, 1 \leq t \leq K, 1 \leq k \leq m$: the total processing time of the class C_t jobs among the first k jobs. We can calculate these quantities in advance.

4. $e_k^{(i-1)t}, 1 \leq i \leq t \leq K+1, 1 \leq k \leq m$: the total processing time of the early jobs among the first k jobs that belong to class C_t and are in I_i.

It is obvious that all the jobs in the first interval $I_1 = [d_0, d_1]$, are early. To make our discussion clearer, we define a series of artificial quantities y_k^{0t}, where $1 \leq t \leq K$ and $1 \leq k \leq m$. They are the total processing times of tardy jobs of class C_t in interval I_1 among the first k assigned jobs. Then $y_k^{0t} = 0, \forall\, 1 \leq t \leq K$ and $1 \leq k \leq m$. It is easy to observe that for any $1 \leq k \leq m$ before deadline d_t, all the C_t jobs are early, and after d_t all the C_t jobs are tardy, while the total processing time of this class is equal to that of the

early jobs plus that of the late jobs. Therefore, we have

$$\left\{\begin{array}{c}\text{total processing}\\ \text{time of the}\\ C_t \text{ jobs}\end{array}\right\} = \left\{\begin{array}{c}\text{total processing time of}\\ \text{the tardy } C_t\text{jobs}\\ \text{from} I_{t+1} \text{to} I_{K+1}\end{array}\right\} + \left\{\begin{array}{c}\text{total processing time of}\\ \text{the early } C_t\text{jobs}\\ \text{from} I_1 \text{to} I_t\end{array}\right\}$$

Mathematically, it can be represented as

$$A_k^t = \sum_{u=t}^{K} y_k^{ut} + \sum_{u=1}^{t} e_k^{ut} \tag{4.1}$$

Then we observe some properties of the optimal schedule by giving lemmas without proofs, since the proofs are similar to the ones for the lemmas given in the next section.

Lemma 3 In the optimal sequence, for any $1 \leq i \leq K$, if straddler S_i is tardy, then for any $1 \leq l \leq i$ and any $i+1 \leq u \leq K$, we have $e_k^{lu} = 0$, in other words, there is no C_u jobs early in interval I_l.

Lemma 4 In the optimal sequence, for any $1 \leq i \leq K-1$, if straddler S_i is early, then $y_k^{iu} = 0$ for all $1 \leq u \leq i$, in other words, there is no tardy jobs in I_{i+1}.

4.2 Structural properties of an optimal schedule

Previously we defined a stop due date problem for the $n - K$ non-straddling jobs. In this section we are going to make same observations on the structural properties of an optimal schedule for the original problem. (From now on, "original problem" will denote the TWT problem with a constant number of distinct due dates.) In any optimal schedule, the machine has clearly no idle time. Hence we can assume that $\sum_{j=1}^{n} p_j > d_K$, since otherwise d_K can be set to ∞ and we have a TWT problem with $K-1$ due dates.

We observe that, in any schedule, each straddler is either early or tardy. Assume

there are M tardy straddlers in an optimal schedule, where $1 \leq M \leq K$. We define an artificial straddler S_0 straddling artificial due date d_0 defined in last section with $w_{S_0} = p_{S_0} = 0$. If we define S_0 as tardy, the indexes of all the tardy straddlers can be numbered as $i_0, i_1, \ldots, i_M$. The corresponding due dates are denoted as $d_{i_0}, d_{i_1}, \ldots, d_{i_M}$. Define $i_{M+1} = \infty$. For any two consecutive tardy indexes i_u, i_{u+1}, where $0 \leq u \leq M$, the segment between two due dates d_{i_u} and $d_{i_{u+1}}$ is defined as a *super-interval*. We denote it as $G_{i_u i_{u+1}}$. Note that between due dates d_{i_u} and $d_{i_{u+1}}$ there is no other tardy straddler. The super-interval $G_{i_u i_{u+1}}$ consists of intervals $I_{i_u+1}, I_{i_u+2}, \ldots, I_{i_{u+1}}$. If $i_{u+1} = i_u + 1$, then $G_{i_u i_{u+1}} \equiv I_{u+1}$. Since S_K is tardy, the last super-interval is $G_{K,K+1} = I_{K+1}$.

With the definition of super-intervals, we have the following results:

Lemma 5 In an optimal schedule, for any tardy straddler S_{i_u} (i_u is the index of tardy straddler) with $1 \leq u \leq M$, for any $1 \leq q \leq i_u$ and $i_u + 1 \leq b \leq K$, we have $e_k^{qb} = 0$.

Proof Suppose that for some $1 \leq \hat{q} \leq i_u$ and $i_u + 1 \leq \hat{b} \leq K$, $e_k^{\hat{q}\hat{b}} > 0$. This implies that there are some $C_{\hat{b}}$ jobs which are early in the interval $I_{\hat{q}}$. Therefore, by exchanging some of the tardy part of S_{i_u} with some or part of these $C_{\hat{b}}$ jobs will reduce the total tardiness, since the tardiness of S_{i_u} is reduced and the $C_{\hat{b}}$ jobs used in the exchange are still early. This is a contradiction of optimality. □

This lemma tells us that, in an optimal sequence, for any class C_b, all the super-intervals before the one where due date d_b lies do not have early C_b jobs.

Lemma 6 In an optimal schedule and for any $1 \leq i \leq K-1$, if S_i is early, then for any $1 \leq u \leq i+1$, we have $y_k^{(i+1)u} = 0$, i.e. there are no late jobs in interval I_{i+1}.

Proof Suppose that there exist $1 \leq \hat{i} \leq K-1$ and $1 \leq \hat{b} \leq \hat{i}+1$, such that $S_{\hat{i}}$ is early, while $y_k^{(\hat{i}+1)\hat{b}} > 0$. This implies that there are some $C_{\hat{b}}$ jobs which are late in I_{i+1}. Then exchanging part of $S_{\hat{i}}$ with some or part of these $C_{\hat{b}}$ jobs will reduce their total tardiness

and S_i is still early.This is a contradiction of optimality. □

A direct consequence of Lemma 6 and the definition of a super-interval is the following.

Lemma 7 **(Bracketing Lemma for early jobs)** Let $1 \leq u \leq M$. In an optimal schedule only jobs from classes C_t, with $i_{u-1} < t \leq i_u$ can be assigned as early in the super-interval $G_{i_{u-1}i_u}$.

Lemma 7 implies that the only non-zero y's are the ones that correspond to the first interval of each super-interval. Therefore, from now on we will use only the values $y_k^{i_u t}$ for any $1 \leq u \leq M$, $1 \leq t \leq i_u$ and $1 \leq k \leq m$. The above lemmas and discussion imply that for every $1 \leq k \leq m$ and for every $1 \leq t \leq K$, such that $i_{s-1} < t \leq i_s$ for some $1 \leq s \leq M$, we have

$$A_k^t = \sum_{\mu=s}^{M} y_k^{i_\mu t} + \sum_{q=i_{s-1}+1}^{t} e_k^{qt} \tag{4.3}$$

If we further observe the early straddlers, we find that

Lemma 8 In an optimal schedule and in any super-interval $[i_u, i_{u+1})$, where $1 \leq u \leq M$, for every $i_u < i < i_{u+1}$ (if there is any), if $S_i \in C_t$, then $i < t \leq i_{u+1}$ must hold.

Proof It is easy to see that $i < t$ since S_i is early. We only need to prove that $t \leq i_{u+1}$. If not, then there exist $i_u \leq i \leq i_{u+1}$ and $S_i \in C_t$ while $t > i_{u+1}$. Exchanging part of S_i with $S_{i_{u+1}}$, we can have less tardiness for $S_{i_{u+1}}$ and S_i is still early. This is a contradiction of the optimality. □

Lemma 8 tells us that we can treat the early straddlers as other early non-straddling jobs. Therefore, tardy straddlers are going to be of particular interest in our algorithm. Since we do not know the tardy straddlers exactly in the optimal schedule, we have to guess them. By guessing we mean the exhaustive enumeration of all possibilities. It includes:

guessing the number $M \leq K$ of tardy straddlers; guessing due dates positions where the tardy straddlers will go; and then guessing jobs as the straddlers (with repetition in the general case where a job can be straddler of more than one due dates) which produces a polynomial number of possibilities since K is a constant. Let $m = n - M$ be the number of the remaining jobs which are ordered according to their weighted shortest processing times (WSPT). With some abuse of terminology, we will call these jobs *non-straddling*, although some of them are the early straddlers.

4.3 A dynamic programming algorithm to find an abstract schedule

In this section, we are going to give a dynamic programming algorithm which finds an *abstract schedule* for the $m = n - M$ non-straddling jobs. An abstract schedule is defined as an assignment of the m non-straddling jobs to the super-intervals so that the following conditions are satisfied:

1. early jobs are feasibly and non-preemptively packed within their assigned super-interval;
2. there is enough empty space so that tardy jobs that are completed in their assigned super-interval can be preemptively packed;
3. there is enough empty space so that the M tardy straddlers can be preemptively packed.

An explanation of the abstract schedule is: all the early jobs are placed non-preemptively but the tardy ones are placed preemptively. Here preemption means the scheduler is allowed to interrupt the processing of a job at any point in time and put a different job on the

machine instead. Specifically the assignments of the tardy jobs are recorded but their actual processes can take place anywhere before the super-intervals assigned. As a result, in an abstract schedule, any tardy job is floating around or before the super-interval that it is assigned, although it is recorded to be finished in that super-interval. An abstract k schedule is an abstract schedule for the first $k \leq m$ non-straddling jobs.

For the specific allocation of the jobs in each super-interval $G_{i_{u-1}i_u}$, we are going to follow the following rules:

1. all the tardy jobs (if there are any) are placed consecutively in WSPT order as a block within the first interval of this super-interval;

2. all the early jobs (if there are any) are placed consecutively in EDD order as a block following the tardy block (if there is any), before due date d_{i_u}.

The dynamic programming (DP) guesses M straddlers. Extending the DP in [Kellerer2006], the states of our DP store the following values for a partial schedule of the first k out of m non-straddling jobs:

$$(k, Z_k, y_k^{i_1 1}, W_k^{i_1 1}, \ldots, y_k^{i_M 1}, W_k^{i_M 1}, \ldots, y_k^{i_1 K}, W_k^{i_1 K}, \ldots, y_k^{i_M K}, W_k^{i_M K}) \qquad (4.4)$$

where Z_k is the total weighted tardiness of the k scheduled jobs. Note that some of the $y_k^{i_u j}$, $W_k^{i_u j}$ may not exist if $i_u < j$ and the weight values $W_k^{i_u j}$ will be needed when the tardy straddlers will be re-inserted at the end. Recall that the k jobs are in WSPT order.

As in [Kellerer2006], to prove that the generated solution is bounded by $1+\epsilon$ of the optimal value, we need to find a lower bound Z_{lb} for all the Z_ks. The property of this lower bound is that twice of it will give an upper bound to Z_k. To make that happen, we start with an obvious lower bound, for instance 1. Then we run the algorithm with $Z^{ub} = 2^x$ for all $x = 0, 1, \ldots, UP$ with UP being a trivial upper bound of OPT, e.g.,

$UP = \log(n^2 w_{max} p_{max}) = O(\log n + \log w_{max} + \log p_{max})$.

The algorithm starts with an initial state $(0, 0, 0, 0, \ldots, 0, 0, \ldots 0, 0)$. The state-to-state transition from state (4.4) corresponds to the insertion of the $k+1$-th job into a super-interval of the partial abstract schedule of the previous k jobs. Such a transition corresponds to the choice of inserting this job in a super-interval, and must be feasible. The feasibility conditions described in detail below require that there is enough empty space to insert the new job in the selected super-interval (non-preemptively as an early job and preemptively as a tardy job), and there is still enough empty space for the re-insertion of the straddlers. Note that the combination of the class C_t of the inserted job and the super-interval $G_{i_u\ 1 i_u}$ chosen for it by the transition determines whether this job is early or tardy: if $t > i_{u-1}$, job J_{k+1} is early; otherwise it is tardy.

The calculation of the empty space in any super-interval is not intuitive. The actual total processing time of the early jobs in this super-interval can be easily computed. But the actual processing time of the tardy jobs in this super-interval is not obvious because parts of them are preempted to the prefix before this super-interval. In addition, the look-like empty space in this super-interval might be occupied by parts of the tardy jobs from future super-intervals. Therefore, we need to define the empty space by considering all these concerns.

Assume now we are considering the empty space in a prefix, i.e., a partial schedule in interval $[d_0, d_l)$.

- L_k^{0l}, $1 \leq l \leq K$, $1 \leq k \leq m$: the total empty space from d_0 to d_l minus the space taken by the jobs whose class indexes are less than or equal to l among the first k jobs.

Based on this definition, in the prefix I_{0l}, there are only up to C_l jobs involved. The total processing time of the $C_1, C_2, \ldots, C_l$ jobs in this prefix is $\sum_{i=1}^{l} A_k^i$. Then we have

$$\sum_{j=1}^{u-1}\sum_{q=i_{j-1}+1}^{i_j}\sum_{b=q}^{i_j} e^{qb} + \sum_{q=i_{u-1}+1}^{l}\sum_{b=q}^{l} e^{qb} + \sum_{j=1}^{u-1}\sum_{b=1}^{i_j} y^{i_j b} + \sum_{j=u}^{M}\sum_{b=1}^{l} y^{i_j b} = \sum_{i=1}^{l} A_k^i$$

where

$$\sum_{j=1}^{u-1}\sum_{q=i_j{}_1+1}^{i_j}\sum_{b=q}^{i_j} e^{qb} = \sum_{q=i_0+1}^{i_1}\sum_{b=q}^{i_1} e^{qb} + \sum_{q=i_1+1}^{i_2}\sum_{b=q}^{i_2} e^{qb} + \cdots + \sum_{q=i_{u-1}+1}^{i_u}\sum_{b=q}^{i_u} e^{qb}$$

Then L_k^{0l} can be computed from the information at hand as follows:

$$\begin{aligned} L_k^{0l} &= d_l - \left(\sum_{j=1}^{u-1}\sum_{q=i_j{}_1+1}^{i_j}\sum_{b=q}^{i_j} e^{qb} + \sum_{q=i_{u-1}+1}^{l}\sum_{b=q}^{l} e^{qb} \right) - \sum_{j=1}^{u-1}\sum_{b=1}^{i_j} y^{i_j b} && (4.5)\\ &= d_l - \left(\sum_{i=1}^{l} A_k^i - \sum_{j=1}^{u-1}\sum_{b=1}^{i_j} y^{i_j b} - \sum_{j=s}^{M}\sum_{b=1}^{l} y^{i_j b} \right) - \sum_{j=1}^{u-1}\sum_{b=1}^{i_j} y^{i_j b} && (4.6)\\ &= d_l - \sum_{i=1}^{l} A_k^i + \sum_{j=u}^{M}\sum_{b=1}^{l} y^{i_j b} && (4.7) \end{aligned}$$

Recall that there are M tardy straddlers $\{S_{i_u}\}_{u=1}^{M}$ overall and each straddler has processing time p_{i_u}. Assume that the $k+1$-th job $J_{k+1} \in C_t$, and consider inserting it into super-interval $G_{i_{u-1}i_u}$. Note that Lemma 3 implies that to even consider such a placement $t \leq i_u$ must hold. Then three feasibility conditions must all be satisfied by a DP transition from state (4.4) with first variable k to state with first variable $k+1$:

Condition 1 If $t \leq i_{u-1}$, i.e., J_{k+1} is tardy

1a if $L_k^{0l} - L_k^{0i_{u-1}} \geq p_{k+1}$ holds, $\forall l$ s.t. $i_{u-1} < l \leq i_u$.

1b if **1a** does not hold, check if $L_k^{0l} > p_{k+1}$ holds $\forall l$, s.t. $i_{u-1} < l \leq i_u$.

1c check if $L_k^{0i_j} > p_{k+1}$ holds $\forall j$, s.t. $u < j \leq M$.

Condition 2 If $i_{u-1} < t \leq i_u$, i.e., J_{k+1} is early

2a check if $L_k^{0l} - L_k^{0i_{u-1}} \geq p_{k+1}$ holds, $\forall l$ s.t. $t \leq l \leq i_u$.

2b if **2a** does not hold, then check if

1. if $\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} \leq L_k^{0i_{u-1}}$: check if $d_l - d_{i_{u-1}} - (\sum_{q=i_{u-1}+1}^{l} \sum_{v=q}^{l} e_k^{qv}) \geq p_{k+1}$ and $L_k^{0l} \geq p_{k+1}$ holds, $\forall l$, s.t. $t \leq l \leq i_u$;
2. if $\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} \geq L_k^{0i_{u-1}}$: check if $L_k^{0l} \geq p_{k+1}$ holds, $\forall l$, s.t. $t \leq l \leq i_u$.

2c check if $L_k^{0i_j} > p_{k+1}$ holds $\forall j$, s.t. $u < j \leq M$.

Condition 3 Check if $L_{k+1}^{0j} \geq \sum_{h=1}^{u-1} p_{i_h}$ holds, $\forall u$, s.t. $1 < u \leq M$ and $\forall j$ s.t. $i_{u-1} < j \leq i_u$.

Condition **(3)** will ensure that there is always enough empty space to fit the straddlers in the final schedule. Condition **(1a)** and **(2a)** are satisfied when there is enough space to fit J_{k+1} as tardy or early in a non-preemptive schedule. We will prove that for a preemptive schedule, Conditions **(2b)** , **(2c)** are enough to guarantee that early jobs can always be inserted non-preemptively, and even if Condition **(1a)** does not hold, as long as Condition **(1b)** and **(1c)** are satisfied we are able to insert tardy jobs preemptively. But we will use the fact that Conditions **(1a)**, **(2a)** and **(3)** are enough for the construction of an optimal DP algorithm which produces an optimal non-preemptive schedule in the analysis of our FPTAS.

There is a another concise way of expressing Condition **(2b)** as shown in Lemma 9 below. We define the empty space in interval $I_{i_{u-1}l}$ where $i_{u-1} < l \leq i_u$ as: the empty space in interval $I_{i_{u-1}l}$ without considering the early jobs which fall into this interval and the class indexes are strictly greater than l. With the information from state (4.4) we have

$L_k^{(i_{u-1})l} := d_l - d_{i_u\ 1} - [\sum_{v=1}^{i_{u-1}} y_k^{(i_{u-1})v} + \sum_{q=i_{u-1}+1}^{l} \sum_{v=q}^{l} e_k^{qv}]$. By substituting equalities (4.3), we have

$$L_k^{i_{u-1}l} = d_l - d_{i_{u-1}} - \left(\sum_{q=i_{u-1}+1}^{l} \sum_{v=q}^{l} e_k^{qv} + \sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} \right) \tag{4.8}$$

Then we have the following result:

Lemma 9 Condition **(2b)** can be replaced by the following: check whether $d_l - d_{i_{u-1}} - (\sum_{q=i_{u-1}+1}^{l} \sum_{v=q}^{l} e_k^{qv} + \max\{\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} - L_k^{0i_{u-1}}, 0\}) \geq p_{k+1}$ holds, $\forall l$, s.t. $t \leq l \leq i_u$.

Proof. If $\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} \leq L_k^{0i_{u-1}}$, then it is obvious that checking $d_l - d_{i_{u-1}} - (\sum_{q=i_{u-1}+1}^{l} \sum_{v=q}^{l} e_k^{qv} + \max\{\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} - L_k^{0i_{u-1}}, 0\}) \geq p_{k+1}$ $\forall l$, s.t. $t \leq l \leq i_u$, is equivalent to checking $d_l - d_{i_{u-1}} - (\sum_{q=i_{u-1}+1}^{l} \sum_{v=q}^{l} e_k^{qv}) \geq p_{k+1}$. If $\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} \geq L_k^{0i_{u-1}}$, then inequality $d_l - d_{i_{u-1}} - (\sum_{q=i_{u-1}+1}^{l} \sum_{v=q}^{l} e_k^{qv} + \max\{\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} - L_k^{0i_{u-1}}, 0\}) \geq p_{k+1}$ is equivalent to $d_l - d_{i_{u-1}} - (\sum_{q=i_{u\ 1}+1}^{l} \sum_{v=q}^{l} e_k^{qv} + \sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} - L_k^{0i_{u-1}}) \geq p_{k+1}$. By exchanging the positions of the terms we have $L_k^{0i_{u-1}} + d_l - d_{i_{u\ 1}} - (\sum_{q=i_{u-1}+1}^{l} \sum_{v=q}^{l} e_k^{qv} + \sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v}) \geq p_{k+1}$. By the definition (4.8), we have $L_k^{0i_{u\ 1}} + L_k^{i_{u\ 1}l} = L_k^{0l} \geq p_{k+1}$ for all $t \leq l \leq i_u$. An interpretation of these inequalities is that all the C_l jobs which are early in the super-interval $G_{i_{u-1}i_u}$ are still early after inserting the new early job J_{k+1}. □

The new state $(k+1, Z_{k+1}, \ldots)$ after the feasible insertion of the $(k+1)$-th job J_{k+1} of class C_t in super-interval $G_{i_{u-1}i_u}$ is computed as follows:

1. J_{k+1} is early: set $y_{k+1}^{i_u j} = y_k^{i_u j}$, $W_{k+1}^{i_u j} = W_k^{i_u j}$ for all $1 \leq u \leq M$, $1 \leq j \leq i_u$. Calculate $Z_{k+1} = Z_k$.

2. J_{k+1} is tardy: set $y_{k+1}^{i_u t} = y_k^{i_u t} + p_{k+1}$, $W_{k+1}^{i_u t} = W_k^{i_u t} + w_{k+1}$. While for all $1 \leq j \leq i_u$ and $j \neq t$, set $y_{k+1}^{i_u j} = y_k^{i_u j}$, $W_{k+1}^{i_u j} = W_k^{i_u j}$. Calculate $Z_{k+1} = Z_k + w_{k+1}(\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} + p_{k+1} + d_{i_{u-1}} - d_t)$. Note that we reject the insertion

if $Z_{k+1} > Z^{ub}$, and 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.

In the abstract schedule, the tardy jobs must complete in their respective super-interval but their processing can take place anywhere before their completion time. The following lemma shows that the tardy jobs can be preemptively inserted.

Lemma 10 Let $u \leq M, 1 \leq k \leq m$. If $L_k^{0i_j} \geq 0, \forall j$ s.t. $1 \leq j \leq u$, then there is enough actual empty space to pack preemptively the tardy jobs that have so far been assigned to the first u super-intervals.

Proof. For a super-interval $G_{i_{u-1}i_u}$, we have $L_k^{i_{j-1}i_j} = L_k^{0i_j} - L_k^{0(i_{j-1})}$ based on the definitions (4.8) and (4.5). When the quantity $L_k^{i_{j-1}i_j}$ is non-negative, it is equal to the empty space in $[d_{i_{j-1}}, d_{i_j})$ plus the space potentially needed in $[d_{i_{j-1}}, d_{i_j})$ by pieces of preempted tardy jobs with completion time after d_{i_j}. It corresponds to a super-interval with an excess of space which can be used to accommodate preempted parts of jobs that complete in the future super-intervals. When the quantity $L_k^{i_{j-1}i_j}$ is negative, it equals the excess space potentially needed by the preempted tardy jobs with completion time in $[d_{i_{j-1}}, d_{i_j})$. It corresponds to a super-interval with an excess portion of tardy jobs which needs to be preempted towards the past. Since $L_k^{0i_h} = \sum_{j=1}^{h} L_k^{i_{j-1}i_j}$, if $L_k^{0i_h} \geq 0, \forall h$ s.t. $1 \leq h \leq u$, then $L_k^{0i_h}$ is the net empty space for accommodating preemptions from jobs that complete after d_{i_j} once all tardy jobs assigned in $[d_0, d_{i_j})$ have been packed. □

The above lemma states that the empty space in prefix I_{0i_u} for all $1 \leq u \leq M$ is the actual free space. In the following lemmas, we show how to feasibly pack the early and tardy jobs in the partial abstract schedule.

Lemma 11 Assume state (4.4) corresponds to an abstract k-schedule. Condition **(2)** and **(3)** imply that job J_{k+1} is packed non-preemptively as early in the intervals $I_{i_{u-1}+1}, \ldots, I_{i_u}$, so that we obtain an abstract $k+1$ schedule. Moreover all early jobs complete as close to their due date as possible.

Proof. We claim that if Condition **(2a)** holds, then job J_{k+1} can be feasibly inserted into interval $L_{i_{u-1}t}$. Since $L_{i_{u-1}t} = L_k^{0t} - L_k^{0i_{u-1}} \geq p_{k+1}$ holds, this means that excluding previously inserted early jobs in this interval whose job class indexes are greater than t, there is enough space for job J_{k+1}. We check the feasibility for the excluded early jobs after they are pushed forward. We check the jobs according to the increasing order of their class indexes. For any $t < l \leq i_u$, since $L_{i_{u-1}l} = L_k^{0l} - L_k^{0i_{u-1}} \geq p_{k+1}$, this means that even if some class C_l jobs are pushed forward because of the insertion of J_{k+1}, they will not be pushed beyond their due date d_l. Also because of the insertion of J_{k+1}, the early C_l jobs are pushed close to their due dates d_l.

If Condition **(2a)** does not hold, that is, any one of the early jobs in the abstract k schedule in super-interval $G_{i_{u-1}i_u}$ becomes tardy because of the insertion of job J_{k+1}, then we consider to move parts of the tardy jobs in this super-interval backwards so that job J_{k+1} can be feasibly inserted. Therefore, Condition **(2b)** needs to be checked. Because of Lemma 10, if $\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} \leq L_k^{0i_{u-1}}$, then this means that the actual empty space before $d_{i_{u-1}}$ is enough for moving all the tardy jobs in super-interval $G_{i_{u-1}i_u}$ to the past. If $\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} \leq L_k^{0i_{u-1}}$, this means that there is only $L_k^{0i_{u-1}} - \sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v}$ processing time of the tardy jobs in super-interval $G_{i_{u-1}i_u}$ can be moved backwards without affecting the the jobs feasibly assigned before $d_{i_{u-1}}$. After parts or all the tardy jobs in super-interval $G_{i_{u-1}i_u}$ are moved to the past, the insertion of the J_{k+1} will not affect the rest of the tardy jobs (if there are any) since it is inserted after the tardy block. Similar to the argument in previous paragraph, the early jobs with class indexes greater than t will not be influenced.

Since Condition **(3)** must hold, space for straddlers is preserved. Since Condition **(2c)** holds (i.e., $L_{k+1}^{0i_j} \geq 0 \; \forall j$, s.t. $u < j \leq M$), by Lemma 8, all the jobs assigned after d_{i_u} are not affected. □

Lemma 12 Assume state (4.4) corresponds to an abstract k-schedule. Conditions **(1)** and **(3)** imply that one can assign job J_{k+1} to complete as tardy in the super-interval $G_{i_{u-1}i_u}$, so that we obtain an abstract $k+1$-schedule.

Proof. If Condition **(1a)** holds, then there is enough space to place job J_{k+1} in interval $I_{i_{u-1},i_{u-1}+1}$ non-preemptively by pushing the early jobs in super-interval $G_{i_{u-1}i_u}$ forward. As argued in Lemma 11, those early jobs will not be pushed beyond their own due dates. If Condition **(1a)** does not hold, Condition **(1b)** can guarantee that J_{k+1} can be preemptively inserted . We know that $L_k^{0l} = L_k^{0i_{u-1}} + L_k^{i_{u-1}l}$ and by Lemma 8, $L_k^{0i_{u-1}}$ is the actual empty space before $d_{i_{u-1}}$. Therefore, if $L_k^{0l} \geq p_{k+1} \; \forall l$ s.t. $i_{u-1} < l \leq i_u$, then job J_{k+1} can use the actual free space in prefix $I_{0i_{u-1}}$ plus the empty space in interval $I_{i_{u-1},i_{u-1}+1}$ by pushing the early jobs with class indexes greater than t forward. The argument for pushing early jobs is similar to the one in Lemma 11. By Lemma 10, we know that the jobs assigned before $d_{i_{u-1}}$ are not affected. At the same time, Condition **(1c)** guarantees that the jobs after d_{i_u} will not be affected and Condition **(3)** guarantees that space for straddlers is preserved. □

4.4 Producing an optimal schedule

When preemption is allowed, the jobs are allocated by the dynamic programming according to the following procedure:

1. the early jobs are placed in their super-interval non-preemptively and as close to their due date as possible;

2. the tardy jobs in the last interval $I_{K,K+1}$ are scheduled in that interval non-preemptively in WSPT order;

3. for $u = M, M-1, \ldots, 1$ look at the tardy jobs with completion times in $G_{i_{u-1}i_u}$, i.e., in interval $I_{i_{u-1},i_{u-1}+1}$ in WSPT order. While there is empty space in this interval, fit in it as much processing time as possible of the job currently under consideration. If at some point there is no more empty space, the rest of the processing times of these tardy jobs will become preempted pieces to be fitted somewhere in $[d_0, d_{i_{u-1}})$. Then, we fill as much of the remaining empty space in $G_{i_{u-1}i_u}$ as possible using preempted pieces belonging to preempted tardy jobs in $[d_{i_u}, d_K]$ in WSPT order (although the particular order does not matter). When we run out of either empty space or preempted pieces, we move to the next $u := u - 1$.

Note that the above process does not change the quantities L_m^{0j}, $j = 1, 2, \ldots, K$, and therefore Condition **(3)** continues to hold.

Next we are going to place the tardy straddlers to complete the schedule. The following lemma shows how to place the straddlers.

Lemma 13 The placement of the tardy straddlers can be done so that the following properties are maintained:

1. straddler S_{i_u} completes at or after d_{i_u} and before $d_{i_{u+1}}$, for all $u = 1, 2, \ldots, M-1$;

2. the prefix of the schedule that contains all straddlers processing time is contiguous, i.e., there are no "holes" of empty space in it.

Proof. We are going to prove this lemma constructively by placing the tardy straddlers one by one. Since Condition **(3)** holds for $u = M$, $j = i_M = K$, we have $L_m^{0i_M} \geq \sum_{h=1}^{M-1} p_{S_{i_h}}$. By Lemma 10, this means there is enough actual empty space before $d_{i_M} = d_K$ for placing

straddlers $S_{i_1}, S_{i_2}, \ldots, S_{i_{M-1}}$. Suppose all the tardy straddlers other than S_{i_M} are already inserted and there is no hole between d_0 and the completion time of the last job before d_{i_M}. Since Condition **(3)** holds for all prefixes, up to now, no violation happens for any job or straddler. We are placing straddler S_{i_M}.

Since all the jobs are going to be placed consecutively, we know that there is $L_m^{0i_M} - \sum_{h=1}^{M-1} p_{S_{i_h}}$ space left before d_{i_u}. If $L_m^{0i_M} - \sum_{h=1}^{M-1} p_{S_{i_h}} \leq p_{S_{i_M}}$, then use the extra empty space $(L_m^{0i_M} - \sum_{h=1}^{M-1} p_{S_{i_h}})$ to fit $(L_m^{0i_M} - \sum_{h=1}^{M-1} p_{S_{i_h}})$ units of $p_{S_{i_M}}$, and fit the remaining $p_{S_{i_M}} - (L_m^{0i_M} - \sum_{h=1}^{M-1} p_{S_{i_h}})$ units right after d_{i_M}. At the same time shift the whole tardy block after d_{i_M} towards the future by the same amount of units. If $L_m^{0i_M} - \sum_{h=1}^{M-1} p_{S_{i_h}} \geq p_{S_{i_M}}$, then start the straddler S_{i_M} at the time $d_{i_M} - (L_m^{0i_M} - \sum_{h=1}^{M-1} p_{S_{i_h}} - p_{S_{i_M}})$ and set the completion time at d_{i_M}. After the insertion of S_{i_M}, if there is any empty space left before d_{i_M}, then just leave it as it is.

Then we place the second straddler $S_{i_{M-1}}$. Again, assume that straddlers $S_{i_1}, \ldots, S_{i_{M-2}}$ have been placed consecutively with other jobs before $d_{i_{M-1}}$. Recall that S_{i_M} starts at $d_{i_M} - (L_m^{0i_M} - \sum_{h=1}^{M-1} p_{S_{i_h}} - p_{S_{i_M}})$. Therefore, we have two cases to consider:

- if in interval $I_{i_{M-1}, i_{M-1}+1}$, early jobs with class indexes greater than $i_{M-1}+1$ can be moved after $d_{i_{M-1}+1}$ without causing any violation (i.e., early job becomes tardy), then there are no preempted pieces of tardy jobs completing after $d_{i_{M-1}}$ in $[d_0, d_{i_{M-1}})$, since these pieces could only have come from the tardy jobs in $G_{i_{M-1}i_M}$, but then it is impossible for that empty space to exist. In this case, because Condition **(3)** holds and by Lemma 10, we know that before $d_{i_{M-1}}$ there are $L_m^{0i_{M-1}} - \sum_{h=1}^{M-2} p_{S_{i_h}}$ units of empty space;

- if in interval $I_{i_{M-1}, i_{M-1}+1}$, there is no early job which can be moved forward (or after $d_{i_{M-1}+1}$), then there are possible preempted tardy jobs before $d_{i_{M-1}}$. In this case, $L_m^{0i_{M-1}} \geq L_m^{0(i_{M-1}+1)}$. Since Condition **(3)** holds, we have $L_m^{0(i_{M-1}+1)} - \sum_{h=1}^{M-2} p_{S_{i_h}} \geq$

$\sum_{h=1}^{M-1} p_{S_{i_h}} - \sum_{h=1}^{M-2} p_{S_{i_h}} = p_{S_{i_{M-1}}}$. By Lemma 10, the empty space before $d_{i_{M-1}}$ is the actual empty space.

In any one of the cases above, straddler $S_{i_{M-1}}$ starts at the time $L_m^{0i_{M-1}} - \sum_{h=1}^{M-2} p_{S_{i_h}}$. It completes either at $d_{i_{M-1}}$ (if $L_m^{0i_{M-1}} \geq \sum_{h=1}^{M-1} p_{S_{i_h}}$) or after $d_{i_{M-1}}$ by amount $\sum_{h=1}^{M-1} p_{S_{i_h}} - L_m^{0i_{M-1}}$. Continue this procedure until all the straddlers are placed correctly. □

We will need property **(2)** in the calculation of the total tardiness of the final schedule below and in our FPTAS. It may force us to preempt straddlers: for example, suppose that the empty space in $[d_0, d_1)$ is much bigger than $\sum_{h=1}^{M} p_{S_{i_h}}$; then our schedule will use $\sum_{h=1}^{M} p_{S_{i_h}}$ units at the beginning of that empty space to process $S_{j_1}, \ldots, S_{j_M}$, while setting their completion times at $d_{j_1}, \ldots, d_{j_M}$ respectively.

After placing all m non-straddling jobs, the dynamic programming will produce a collection of states with their first variable equal to m. Since all these states are feasible, Lemma 13 implies that we can re-insert the straddlers at their correct position without causing early jobs to be tardy and affecting the placement in intervals of the tardy non-straddling jobs, thus creating a number of candidate complete schedules. Let $\{T_{i_u}\}_{u=1}^{M}$ be the tardiness of the M tardy straddlers. Define the part of S_{i_u} that completed after due date d_{i_u} as x_{i_u}, then

$$x_{i_u} := \max\{0, \sum_{h=1}^{u} p_{S_{i_h}} - L_m^{0i_u}\} \tag{4.9}$$

Assume $S_{i_u} \in C_t$ with $t \leq i_u$, we have

$$T_{i_u} = x_{i_u} + d_{i_u} - d_t, \forall u = 1, \ldots, M$$

and the total weighted tardiness of candidate schedule is

$$Z = Z_m + \sum_{u=1}^{M} w_{i_u} T_{i_u} + \sum_{u=1}^{M} (\sum_{j=1}^{i_u} W_m^{i_u j}) x_{i_u} \tag{4.10}$$

The algorithm outputs a schedule with minimum Z by tracing back the feasible transitions, starting from the state that has the Z_m which produced the minimum Z.

Theorem 1 The dynamic programming algorithm above produces an optimal schedule.

Proof. We prove that our dynamic programming algorithm contains a schedule which is the same as an optimal non-preemptive schedule, that is, if a job in an optimal schedule is early in some super-interval, then our DP also finds it early in that super-interval; if a job in an optimal schedule is tardy in some super-interval, then our DP also finds it tardy preemptively in that super-interval.

Take any optimal non-preemptive schedule (which we already know that exists) and remove the straddlers, say there are K of them. For the rest $m = n - K$ non-straddling jobs in that schedule, we prove that they satisfy the feasibility conditions of our DP by induction. For job J_1, Conditions **(1a)**, **(2a)** and **(3)** hold obviously since it is inserted non-preemptively . Assume up to the placement of job J_k, Conditions **(1a)**, **(2a)** and **(3)** hold. We show that they also hold for the placement of job J_{k+1} in the super-interval prescribed by the optimal schedule.

It is clear that Condition **(3)** is true for the whole sequence (since the straddlers were correctly placed in the schedule).

Assume job $J_{k+1} \in C_t$ is inserted early in $G_{i_{u-1}i_u}$ in the optimal schedule. Since jobs $J_1, \ldots, J_k$ are inserted non-preemptively, for any $t \leq l \leq i_u$, the total processing time of the tardy jobs and early jobs with class indexes less than or equal to l placed in interval $I_{i_{u-1}l}$ is $\sum_{v=1}^{i_u-1} y_k^{i_{u-1}v} + \sum_{l}^{v=i_u+1} \sum_{q=i_u+1}^{v} e_k^{qv}$. Since job J_{k+1} is non-preemptively placed in super-interval $G_{i_{u-1}i_u}$ (more specifically in interval $I_{i_u\ t}$ because $t \leq l$), we have that

$$\sum_{v=1}^{i_u-1} y_k^{i_u\ 1v} + \sum_{l}^{v=i_u+1} \sum_{q=i_u+1}^{v} e_k^{qv} + p_{k+1} \leq d_l - d_{i_{u-1}} \tag{4.11}$$

By the definitions of L_k^{0l} and $L_k^{0i_u\ 1}$, inequality (4.11) can be written as $L_k^{0l} - L_k^{0i_u\ 1} \geq p_{k+1}$, $\forall l$ s.t. $t \leq l \leq i_u$. Therefore Condition **(2a)** holds for the placement of job J_{k+1}. Similarly we can show that Condition **(1a)** holds when job J_{k+1} is tardy in its super-interval in the optimal schedule.

Therefore there is a path in the DP transition diagram that corresponds to the placement of jobs according to the given optimal non-preemptive schedule, hence the final schedule produced by the algorithm has optimal tardiness. □

If we do not allow preemption, then the proof of the above theorem goes through without checking for Conditions **(1c)**, **(2c)**, since they are satisfied trivially by the optimal non-preemptive schedule.

Corollary 1 The non-preemptive DP algorithm with feasible transitions restricted to only those that satisfy Conditions **(1a)**, **(2a)** and **(3)** still produces an optimal non-preemptive schedule.

For the proof of the approximation ratio guarantee below, we will compare the solution produced by our FPTAS to the optimal schedule of the corollary.

4.5 The FPTAS

The transformation of the pseudo-polynomial algorithm into an FPTAS follows closely Algorithm **Eps** in [Kellerer2006] . In the following we use the term DP to refer to the entire process, since the running time of the dynamic programming part dominates the total running time.

4.5.1 The algorithm and the complexity

For any $\epsilon > 0$, we guess an upper-bound (as discussed in previous section) Z^{ub} such that $Z_{lb} := \frac{Z^{ub}}{2} \leq OPT \leq Z^{ub}$. Define $\delta = \frac{\epsilon Z_{lb}}{4m}$. Consider a state

$$(k, Z_k^*, y_k^{i_1 1*}, W_k^{i_1 1*}, \ldots, y_k^{i_M 1*}, W_k^{i_M 1*}, \ldots, y_k^{i_1 K*}, W_k^{i_1 K*}, \ldots, y_k^{i_M K*}, W_k^{i_M K*})$$

of the exact dynamic programming.

From this state, we will deduce the states

$$(k, Z_k, y_k^{i_1 1}, W_k^{i_1 1}, \ldots, y_k^{i_M 1}, W_k^{i_M 1}, \ldots, y_k^{i_1 K}, W_k^{i_1 K}, \ldots, y_k^{i_M K}, W_k^{i_M K})$$

used by the FPTAS dynamic programming as follow:

- Round variable Z_k to the next multiple of δ. Hence Z_k takes at most $\frac{Z^{ub}}{\delta} = \mathrm{O}(\frac{n}{\epsilon})$ number of distinct values.

- For all $1 \leq u \leq M$, round $W_k^{i_u j}$ to the nearest power of $(1+\epsilon/2)^{1/m}$ so that at most $\log_{(1+\epsilon/2)^{1/m}} W = \mathrm{O}(n \log W)$ number of distinct values are considered, where W is the total weight of the n jobs;

- The rounding of $y_k^{i_u j}$ for all $1 \leq u \leq M$ is more complicated than the above rules. Assume there are N distinct weight values among the non-straddling jobs and sort them in decreasing order $w_{\pi(1)} > w_{\pi(2)} > \cdots > w_{\pi(N)}$. It is easy to see that any $y_k^{i_u j}$ is bounded by $\frac{Z^{ub}}{w_{\pi(N)}}$. Define a division of the time interval $[0, \frac{Z^{ub}}{w_{\pi(N)}}]$ as $\{\hat{H}_{i'} := [\frac{Z^{ub}}{w_{\pi(i'-1)}}, \frac{Z^{ub}}{w_{\pi(i')}}]\}_{i'=1}^{N}$. In turn, for each interval $\hat{H}_{i'}$, we divide it into subintervals $\{\hat{I}_{i'j'}(i)\}_{j'=1}^{x_i^{i'}}$ of length $\delta_i = \frac{\delta}{i w_{\pi(i')}}$ for all $1 \leq k \leq m$, $1 \leq i \leq K$ and $1 \leq j \leq i$. Here $x_i^{i'} = \lceil \frac{\frac{Z^{ub}}{w_{\pi(i')}} - \frac{Z^{ub}}{w_{\pi(i'-1)}}}{\delta_i} \rceil$ is the number of subintervals, although the length of the last subinterval may be less than δ_i. For each state $(k, Z_k, y_k^{i_1 1}, W_k^{i_1 1}, \ldots, y_k^{i_M K}, W_k^{i_M K})$,

the dynamic programming applies $O(K)$ transitions to generate new states $(k+1, Z_{k+1}, y_{k+1}^{i_1 1}, W_{k+1}^{i_1 1}, \ldots, y_{k+1}^{i_M K}, W_{k+1}^{i_M K})$. For the set of states which have the same values of $Z_{k+1}, W_{k+1}^{i_1 1}, \cdots, W_{k+1}^{i_M K}$, we round $y_{k+1}^{i_u j}$ in the following way: group all the $y_{k+1}^{i_u j}$ values that fall into the same subinterval $\hat{H}_{i'j'}$ together and keep only the smallest and the largest values in this group, say $y_{k+1}^{i_u j^{max}}$ and $y_{k+1}^{i_u j^{min}}$. We emphasize that these two values correspond to the actual processing times of two sets of tardy jobs, and therefore none of these two values is greater than A_{k+1}^{j}. Hence from the group of states generated by the DP transition, we produce and store states with at most two values at position $y_{k+1}^{i_u j}$, that is, $(k+1, Z_{k+1}, y_{k+1}^{i_1 1}, W_{k+1}^{i_1 1}, \ldots, y_{k+1}^{i_u j^{min}} \ldots, y_{k+1}^{i_M K}, W_{k+1}^{i_M K})$ and $(k+1, Z_{k+1}, y_{k+1}^{i_1 1}, W_{k+1}^{i_1 1}, \ldots, y_{k+1}^{i_u j^{max}}, \ldots y_{k+1}^{i_M K}, W_{k+1}^{i_M K})$.

Lemma 14 The algorithm runs in time $\mathbf{O}(\epsilon^{-1} n \log W \log P)^{\Theta(K^2)}$.

Proof. In the worst case $M = K$, for each of the $\frac{K(K+1)}{2}$ positions of $y_{k+1}^{i_u j}$, we have at most

$$\begin{aligned} & \frac{\frac{Z^{ub}}{w_{\pi(1)}}}{\delta_i} + \sum_{i=2}^{h} \frac{(\frac{Z^{ub}}{w_{\pi(i)}} - \frac{Z^{ub}}{w_{\pi(i-1)}})}{\delta_i} \\ = \; & i\frac{Z^{ub}}{w_{\pi(1)}}\frac{w_{\pi(1)}}{\delta} + i\sum_{i=2}^{h}(\frac{Z^{ub}}{w_{\pi(i)}} - \frac{Z^{ub}}{w_{\pi(i-1)}})\frac{w_{\pi(i)}}{\delta} \\ = \; & \mathbf{O}(\frac{n^2}{\epsilon}) \end{aligned}$$

distinct subintervals, or $\mathbf{O}((\frac{n^2}{\epsilon})^{\frac{K(K+1)}{2}})$ combinations of subintervals. When the combination of the subintervals is fixed, we have $2^{\frac{K(K+1)}{2}}$ number of combinations for $y_{k+1}^{i_u j}$'s, since there are only two choices for each of them, the maximum and minimum values. Therefore, for the same values of $Z_{k+1}, W_{k+1}^{i_1 1}, \cdots, W_{k+1}^{i_M K}$ we save $\mathbf{O}((\frac{n^2}{\epsilon})^{\frac{K(K+1)}{2}} 2^{\frac{K(K+1)}{2}}) = \mathbf{O}(\epsilon^{-\frac{K(K+1)}{2}} n^{K(K+1)})$ number of states. Taking into account the other variables of the state and the initial guessing for the tardy straddlers and upper-bounds, the overall complexity of the algorithm is $\mathbf{O}(\epsilon^{-1} n \log W \log P)^{\Theta(K^2)}$. □

Assume that states $(k, Z_k^*, y_k^{i_1 1*}, W_k^{i_1 1*}, \ldots, y_k^{i_M 1*}, W_k^{i_M 1*}, \ldots, y_k^{i_M K*}, W_k^{i_M K*})$ with

$k = 0, 1, \ldots, m$ form an optimal sequence of transitions in the dynamic programming that produces an optimal schedule without preemption. In the next section, we are going to show that our rounding algorithm finds a sequence of states $(k, Z_k, y_k^{i_1 1}, W_k^{i_1 1}, \ldots, y_k^{i_M K}, W_k^{i_M K})$ with $k = 0, 1, \ldots, m$ whose transitions from one state to the next match exactly the job placement decisions of the optimal schedule generated by the DP. We can prove this because we take advantage of the preemption of the tardy jobs. Although our algorithm overestimates the space needed by the tardy jobs by rounding up variables y's, the corresponding space of the early jobs (of the same job class) actually is decreased since the total space of the early and tardy jobs of the same class does not change. By allowing preemption on tardy jobs, we can place the overestimated processing time of the tardy jobs in the places of the early jobs whose total processing time is reduced by an equal amount (this amount of space may be divided into several super-intervals though).

4.5.2 Proof of near optimality

We are going to show that among the schedules that generated by the rounding algorithm, an optimal schedule is included and a schedule (might be different from the previous one) which produces an objective value bounded by $(1 + \epsilon)OPT$ is also included.

Lemma 15 Our rounding algorithm contains a sequence of transitions that is the same as an optimal sequence of transition found by exact the DP. Here "the same" means identical job placements.

Proof Another way to present this lemma is that for any one of the m jobs (in WSPT order), if its placement is feasible in an optimal schedule, then it can also be feasibly found by our rounding algorithm. We prove this lemma by way of induction. If a placement of the first job J_1 is feasible for the exact DP, then it is obvious that this job is feasible for

our rounding algorithm. Assume that up to job J_k our rounding algorithm finds identical placements as the exact DP does in an optimal sequence. We look into the placement of job J_{k+1}. Suppose that $J_{k+1} \in C_t$ and its placement is in super-interval $G_{i_{u-1}i_u}$ in the optimal schedule. There are two cases to consider according to the optimal placement:

Case 1: J_{k+1} **is early.** Since J_{k+1} is early , $i_{u-1} < t \leq i_u$ and $L_k^{0l*} - L_k^{0i_{u-1}*} \geq p_{k+1}$ holds in the optimal schedule, $\forall l$ s.t. $t \leq l \leq i_u$. Therefore we have $L_k^{0l*} \geq L_k^{0l*} - L_k^{0i_{u-1}*} \geq p_{k+1}$. Additionally, it is trivial to see that $L_k^{0\hat{l}} \geq L_k^{0i_u*} \geq p_{k+1}$ holds $\forall \hat{l}$, s.t. $i_u < \hat{l} \leq K$. In our algorithm, if $L_k^{0l} - L_k^{0i_{u-1}} \geq p_{k+1}$ holds, $\forall l$ s.t. $t \leq l \leq i_u$, then condition **(2a)** is satisfied. Otherwise, we need to check the two cases of condition **(2b)**:

1. if $\sum_{v=1}^{i_{u-1}} y_k^{i_u\ 1v} \leq L_k^{0i_u\ 1}$: since $\sum_{q=i_{u-1}+1}^{l} \sum_{v=q}^{l} e_k^{qv}$ is rounded down, we have $d_l - d_{i_{u-1}} - (\sum_{q=i_{u-1}+1}^{l} \sum_{v=q}^{l} e_k^{qv}) \geq d_l - d_{i_{u-1}} - (\sum_{q=i_{u-1}+1}^{l} \sum_{v=q}^{l} e_k^{qv*}) \geq L_k^{0l*} - L_k^{0i_{u-1}*} \geq p_{k+1}$;
2. if $\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} \geq L_k^{0i_{u-1}}$: since $L_k^{0l} \geq L_k^{0l*}$, we have $L_k^{0l} \geq p_{k+1}$. Also, since $L_k^{0i_j*} \geq p_{k+1}$ holds $\forall j$, s.t. $u < j \leq M$, we have $L_k^{0i_j} \geq L_k^{0i_j*} \geq p_{k+1}$.

Additionally, Condition **(2c)** is satisfied, because $L_k^{0i_j} \geq L_k^{0i_j*} \geq p_{k+1}$ for all $u < j \leq M$. Hence J_{k+1} can be placed early in super-interval $G_{i_{u-1}i_u}$ by the rounding algorithm. (Note that, for the exact DP, we know that the empty space in the prefix super-interval is increasing, that is, $L_k^{0i_s} \leq L_k^{0i_j}$ if $1 \leq s \leq j \leq M$. But this does not hold in FPTAS because of preemption. However we have the property that $L_k^{0l} \geq L_k^{0l*}$ $\forall l$, s.t. $1 \leq l \leq K$.)

Case 2: J_{k+1} **is tardy** Since J_{k+1} is tardy, $t \leq i_{u-1}$ and $L_k^{0l*} - L_k^{0i_{u-1}*} \geq p_{k+1}$ holds, $\forall l$ s.t. $t \leq l \leq i_u$. Therefore we have $L_k^{0l*} \geq L_k^{0l*} - L_k^{0i_{u-1}*} \geq p_{k+1}$. In addition, it is trivial to see that $L_k^{0\hat{l}} \geq L_k^{0i_u*} \geq p_{k+1}$ holds $\forall \hat{l}$, s.t. $i_u < \hat{l} \leq K$. In the rounding algorithm,

if $L_k^{0l} - L_k^{0i_u\ 1} \geq p_{k+1}$ holds, $\forall l$ s.t. $t \leq l \leq i_u$, then the proof is trivial. Otherwise we have $L_k^{0l} \geq L_k^{0l*} \geq L_k^{0l*} - L_k^{0i_{u-1}*} \geq p_{k+1}$ $\forall l$, s.t. $i_{u-1} < l \leq i_u$. Further we have $L_k^{0i_j} \geq L_k^{0i_j*} \geq p_{k+1}$holds $\forall j$, s.t. $u < j \leq M$. Hence Condition **(1c)** is satisfied. J_{k+1} can be placed tardy by the rounding algorithm.

□

In the rest of the thesis, we work with these two special sequences and their transitions. We observe that $L_m^{0j*} \geq \sum_{h=1}^{u-1} p_{S_{i_h}}$, $\forall j, u$ s.t. $1 < u \leq M$ and $i_{u-1} < j \leq i_u$ from Condition **(3)**, which is satisfied by the optimal DP. Moveover, $L_m^{0l} \geq L_m^{0l*}$ $\forall l$ s.t. $1 \leq l \leq K$ (as proved in Lemma 15). Hence $L_m^{0j} \geq \sum_{h=1}^{u-1} p_{S_{i_h}}$, $\forall j, u$ s.t. $1 < u \leq M$ and $i_{u-1} < j \leq i_u$, i.e., Condition **(3)** is satisfied by the last state produced by our algorithm in the sequence of transitions we study, and therefore we can feasibly complete the schedule produced in this way with the insertion of the tardy straddlers. We prove the approximation ratio guarantee for the schedule produced by our algorithm, by proving this guarantee when the special transition sequence above is followed. We emphasize that our algorithm may not output the schedule corresponding to that sequence, since its approximate estimation of the total tardiness may lead it to picking another one, with a smaller estimate of the total tardiness. For each $1 \leq k \leq m$ and $1 \leq u \leq M$, define $B_{iu}^*(k) := \max\{w_h \mid k \leq h \leq m, y_h^{i_u j} \neq 0, 1 \leq j \leq i_u\}$ and if no job is tardy in super-interval $G_{i_{u-1}i_u}$, set $B_{i_u}^*(k) := 0$.

Lemma 16 For every $1 \leq k \leq m$, $1 \leq u \leq M$ and $1 \leq j \leq i_u \leq K$, we have

$$Z_k \leq Z_k^* + 2k\delta; \tag{4.12}$$

$$0 \leq i_u B_{i_u}^*(k)(y_k^{i_u j} - y_k^{i_u j*}) \leq \delta, \tag{4.14}$$

Proof We are going to prove this lemma by induction. Essentially the proof is the same as the one of lemma 1 given in [Kellerer2006], but we include it here for completeness. Assume that $J_{k+1} \in C_t$ with $1 \leq t \leq K$ and is to be inserted in super-interval $G_{i_u i_{u+1}}$. If J_{k+1} is early, then $Z^*_{k+1} = Z^*_k$; if it is tardy, then $Z^*_{k+1} = Z^*_k + w_{k+1}(\sum_{j=1}^{i_u} y_k^{i_u j^*} + p_{k+1} + d_{i_u} - d_t)$. Define function $\phi_{i_u t}(y_k^{i_u t}) = y_k^{i_u t} + p_{k+1}$ if J_{k+1} is tardy in $G_{i_u i_{u+1}}$ and $\phi_{i_u t}(y_k^{i_u t}) = y_k^{i_u t}$ otherwise. Denote the rounded value of $\phi_{i_u t}(y_k^{i_u t})$ as $y_{k+1}^{i_u t}$.

For $k = 1$, the proof is trivial by the description of the ways of rounding. Assume that the lemma is true for $k = s < m$ and we are going to show that the lemma is true for $k = s + 1$. If in the optimal sequence, $y_{s+1}^{i_u t*} = y_s^{i_u t*} + p_{s+1} \neq 0$, then J_{s+1} is tardy in $G_{i_u i_{u+1}}$ and hence $B^*_{i_u}(s+1) > 0$. Assume $B^*_{i_u}(s+1) = w_v$ where $v \geq s+1$, we have $B^*_{i_u}(s+1) y_{s+1}^{i_u t*} = w_v y_{s+1}^{i_u t*} \leq w_v y_v^{i_u t*} \leq Z^{ub}$ since $y_k^{i_u t*}$ is increasing with the increase of k. Then we have $y_{s+1}^{i_u t*} \leq \frac{Z^{ub}}{B^*_{i_u}(s+1)}$. This implies that $y_{s+1}^{i_u t*}$ belongs to some subinterval of length at most $\frac{\delta}{i_u B^*_{i_u}(s+1)}$.

We have shown in Lemma 15 that whatever the exact DP does to job J_{s+1} in an optimal schedule, our rounding algorithm can find the same placement, so we have

$$0 \leq \phi_{i_u t}(y_s^{i_u t}) - y_{s+1}^{i_u t*} = y_s^{i_u t} - y_s^{i_u t*} \leq \frac{\delta}{i_u B^*_{i_u}(s)} \leq \frac{\delta}{i_u B^*_{i_u}(s+1)}$$

since $B^*_{i_u}(s) \geq B^*_{i_u}(s+1)$. Therefore $\phi_{i_u t}(y_s^{i_u t})$ and $y_{s+1}^{i_u t*}$ are either in the same subinterval or in two consecutive subintervals. If the first case is true, the largest value in that interval is picked as the rounded value $y_{s+1}^{i_u t}$; if the second is true, the smallest value in the next subinterval is picked as the rounded value. Thus we have the result (4.14).

Now we are back to prove (4.12). If J_{s+1} is inserted early, the result is trivial. If it is

tardy, We have

$$\begin{aligned} Z_{s+1} &\leq Z_s + w_{s+1}(\textstyle\sum_{j=1}^{i_u} y_s^{i_u j} + + p_{s+1} + d_{i_u} - d_t) + \delta \\ &\leq Z_s^* + 2s\delta + w_{s+1}(\textstyle\sum_{j=1}^{i_u}(y_s^{i_u j^*} + \frac{\delta}{i_u B_{i_u}^*(s+1)}) + p_{s+1} + d_{i_u} - d_t) + \delta \\ &= Z_s^* + w_{s+1}(\textstyle\sum_{j=1}^{i_u} y_s^{i_u j^*} p_{s+1} + d_{i_u} - d_t) + i_u w_{s+1}\frac{\delta}{i_u B_{i_u}^*(s+1)} + \delta + 2s\delta \\ &\leq Z_{s+1}^* + 2(s+1)\delta \end{aligned}$$

where the first inequality takes into account the increase of Z_{s+1} by at most δ due to its rounding and the last inequality is due to the optimal DP transition for J_{s+1}. □

The following theorem proves that the proposed polynomial algorithm is an FPTAS. Its proof is an extension of the proof of Lemma 2 in [Kellerer2006], and we include it here for completeness purposes.

Theorem 2 Assume that the exact DP finds a chain of states which gives an optimal function value Z^* after inserting the straddlers back. Then for any $\epsilon > 0$, the rounding algorithm outputs a schedule with objective value Z satisfying $Z \leq (1+\epsilon)Z^*$.

Proof To show that function value generated by the rounding algorithm is of ratio $1+\epsilon$, we need firstly to show that $W_m^{i_u j*} \leq W_m^{i_u j} \leq (1+\frac{\epsilon}{2})W_m^{i_u j*}$, where $1 \leq u \leq M$, $1 \leq j \leq i_u$. Since the rounding of all these W's are the same, we can just prove any one of them.

To prove the above inequality, we need to show that $W_k^{i_u j} \leq (1+\frac{\epsilon}{2})^{k/(m)} W_k^{ij*}$ by induction on $1 \leq k \leq m$. For $k = 1$, it is trivial. Assume the inequality holds for k, we check the truth of the inequality for $k+1$. If job J_{k+1} is inserted early, then the result is also trivial since the values of W's remain the same; if inserted tardy, then $W_k^{i_u j} + w_{k+1}$ is

rounded to $W_{k+1}^{i_u j}$, by the definition of rounding, we have

$$\begin{aligned} W_{k+1}^{i_u j} &\leq (W_k^{i_u j} + w_{k+1})(1+\tfrac{\epsilon}{2})^{1/m} \\ &\leq [W_k^{i_u j*}(1+\tfrac{\epsilon}{2})^{k/m} + w_{k+1}](1+\tfrac{\epsilon}{2})^{1/m} \\ &\leq W_k^{i_u j*}(1+\tfrac{\epsilon}{2})^{(k+1)/m} + w_{k+1}(1+\tfrac{\epsilon}{2})^{(k+1)/m} \\ &= (W_k^{i_u j*} + w_{k+1})(1+\tfrac{\epsilon}{2})^{(k+1)/m} \\ &= W_{k+1}^{i_u j*}(1+\tfrac{\epsilon}{2})^{(k+1)/m} \end{aligned}$$

When $k+1=m$, we have $W_m^{i_u j*} \leq W_m^{i_u j} \leq (1+\frac{\epsilon}{2}) W_m^{i_u j*}$.

Let Z_m be the function value of the feasible sequence found by the algorithm before inserting straddlers. Recall from (4.9) that $x_{i_u}^* := \max\{0, \sum_{h=1}^{u} p_{S_{i_h}}^* - L_m^{0 i_u *}\}$. Since $L_m^{0 i_u *}$ is rounded up, $x_{i_u}^*$ is rounded down or becomes 0. Therefore we have

$$\begin{aligned} Z &= Z_m + \textstyle\sum_{u=1}^{M} w_{i_u} T_{i_u} + \sum_{u=1}^{M} (\sum_{l=1}^{i_u} W_m^{i_u l}) x_{i_u} \\ &= Z_m + \textstyle\sum_{u=1}^{M} w_{i_u} (x_{i_u} + d_{i_u} - d_t) + \sum_{u=1}^{M} (\sum_{l=1}^{i_u} W_m^{i_u l}) x_{i_u} \\ &\leq Z_m + \textstyle\sum_{u=1}^{M} w_{i_u} (x_{i_u}^* + d_{i_u} - d_t) + \sum_{u=1}^{M} (\sum_{l=1}^{i_u} W_m^{i_u l}) x_{i_u}^* \\ &\leq Z_m^* + 2m\delta + \textstyle\sum_{u=1}^{M} w_{i_u} (x_{i_u}^* + d_{i_u} - d_t) + (1+\frac{\epsilon}{2}) \sum_{u=1}^{M} (\sum_{l=1}^{i_u} W_m^{i_u l *}) x_{i_u}^* \\ &\leq Z_m^* + \tfrac{\epsilon}{2} Z_{lb} + \textstyle\sum_{u=1}^{M} w_{i_u} (x_{i_u}^* + d_{i_u} - d_t) + \sum_{u=1}^{M} (\sum_{l=1}^{i_u} W_m^{i_u l *}) x_{i_u}^* \\ &\quad + \tfrac{\epsilon}{2} \textstyle\sum_{u=1}^{M} (\sum_{l=1}^{i_u} W_m^{i_u l *}) x_{i_u}^* \\ &\leq Z^* + \tfrac{\epsilon}{2} Z^* + \tfrac{\epsilon}{2} Z^* \\ &= (1+\varepsilon) Z^* \end{aligned} \tag{4.16}$$

□

The algorithm can be easily extended to the cases where some straddler spans more than two due dates. If there is one straddler which spans over more than two due dates and it is early, then our algorithm can place it correctly because it is treated as an early job. If there is more than one tardy straddler that cross more than two due dates, then our algorithm

will apply exhaustive enumeration on the tardy straddlers. For example, assume there is one straddler which crosses more than two due dates, denoted as S_{span}. Then the algorithm guesses the number of consecutive due dates which correspond to straddler S_{span}; then it guesses the due dates positions for S_{span}; thirdly it guesses a job as S_{span} . If there are more than one such kind of straddlers, then we need to guess the number of such kind of straddlers and for each of them apply the guessing above.

Chapter 5

Conclusions and future work

In this thesis, we solve the total weighted tardiness problem with a fixed number of distinct due dates by giving a fully polynomial time approximation scheme (FPTAS).

We observe that only the placement of tardy straddlers is critical. Therefore the time intervals between consecutive tardy straddlers form the basic "super-interval" units on the time horizon. By exhaustively enumerating the jobs and due dates, we can guess the tardy straddlers but we place them only when all the non-straddling jobs are placed on the time line. For the each of the non-straddling jobs, we can guess its placement as either early or tardy. Depending on its due date, if a job is decided to be tardy, it can be placed in any super-interval that after its due date. If this job is decided to be early, then it can only be scheduled in one of super-intervals that before its due date. This helps to shrink the state space of the dynamic program. In each super-interval, if there are tardy jobs, the first tardy job starts from the left most due date; otherwise the first early job starts from there.

When it comes down to the algorithm for non-straddling job, it is performed in two stages. Firstly, via the dynamic programming with preemption, we compute an assignment of the job completion times to the time horizon where only a subset of the jobs is explicitly packed and the rest are left "floating" from their completion time backwards (preempted).

Secondly, applying the rounding schemes to the newly calculated value of each variable and we store only a polynomial number of states. Since all the feasible conditions make the algorithm only produce feasible schedules, for each of them, we insert the straddlers back one by one. To insert each straddler, we need to push the job block in its right hand super-interval toward right such that the straddler can be exactly inserted. If up to some straddler, it can be inserted without pushing the job block at all, then we simply disregard that feasible schedule.

Two crucial properties that both the algorithm in [Kellerer2006] and algorithm in [Karakostas2009] have can be summarized as the following: first of all, the step-by-step mimicking of the optimal chain of computation is crucial for bounding the approximation error, although the schedule we output may be sub-optimal due to the rounded estimation of weighted tardiness; second of all, the rounding scheme produces values which correspond to actual schedules. One big advantage of our algorithm over Kellerer's is the implementation of "preemption" which makes our extension from one due date to a constant number due dates possible. Additionally, observing the structural properties of an optimal schedule plays a very critical role in helping us find the FPTAS.

To complete this thesis, it is necessary to indicate that the approximability of the total weighted tardiness problem with arbitrary number of distinct due dates remains open. Therefore, a future work is to design an FPTAS (if there is) for the TWT problem with arbitrary number of distinct due dates.

Bibliography

[Abdul-Razacq1990] T.S. Abdul-Razacq, C.N. Potts, L.N. Van Wassenhove, "A survey of algorithms for the single machine total weighted tardiness scheduling problem." *Discrete Applied Mathematics,* 26 (1990) 235-253.

[Arkin1991] E.M. Arkin, R.O. Roundy, "Weighted tardiness scheduling on parallel machines with proportional weights." *Operaitons Research,* 39 (1991) 64-81.

[Akturk1998] M.S. Akturk, M.B. Yildirim "A new lower bounding scheme for the total weighted tardiness problem." *Computers and Operations Research,* 25 (1998) 265-278.

[Babu2004] P. Babu, L. Peridy, E. Pinson, "A branch and bound algorithm to minimize total weighted tardiness on a single processor." *Annals of Operations Research,* 129 (2004) 33-46.

[Baker1974] K.R. Baker, "Introduction to Sequencing and Scheduling." Wiley, NY, 1974.

[Besten2000] M.L. den Besten, T. Stützle, M. Dorigo, "An ant colony optimization application to the single machine total weighted tardiness problem." *Abstract Proceedings for ANTS' 2000: From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms,* Brussels, Belgium, September (2000) 39-42.

[Borgulya2002] I. Borgulya, "A cluster-based evolutionary algorithm for the single machine total weighted tardiness-shceduling problem." *Journal of Computing and Information Technology,* 10 (2002) 211-217.

[Cheng2005] T.C.E. Cheng, C.T. Ng, J.J. Yuan and Z.h. Liu, "Single machine scheduling to minimize total weighted tardiness." *European Journal Operations Research,* 165 (2005) 423-443.

[Congram2002] R.K. Congram, C.N. Potts, S.L. van de Velde, "An iterated dynaserch algorithms for the single-machine total weighted tardiness scheduling problem." *INFORMS Journal on Computing,* 14 (2002) 52-67.

[Conway1967] R.W. Conway, W.L. Maxwell and L.W. Miller, "Theory of Scheduling." Addison-Wesley Publishing Co., MA, 1967.

[Crauwels1998] H.A.J. Crauwels, C.N. Potts, L.N. Van Wassenhove, "Local search heuristics for the single machine total weighted tardiness scheduling problem." *INFORMS Journal on Computing,* 10 (1998) 341-350.

[Du1990] J. Du and J.Y.-T. Leung, "Minimizing total tardiness on one machine is NP-hard." *Mathematics of Operations Resarch,* 15 (3) (1990) 483-495.

[Elmaghraby1968] S.E. Elmaghraby, "The one machine sequencing problem with delay costs." *Journal of Industrial Engineering,* 19 (1968) 105-108.

[Emmons1969] H. Emmons, "One-maching sequencing to minimize certain functions of job tardiness." *Operations Research,* 17 (1969) 701-715.

[Fathi1990] Y. Fathi, H. W. L. Nuttle, "Heuristics for the common due date weighted tardiness problem." *IIE Transactions,* 22 (3) (1990) 215-225.

[Hoogeveen1995] J.A. Hoogeveen, S.L. Van de Velde, "Stronger Lagrangian bounds by use of slack variables: Applications to machine scheduling problems." *Mathematical Programming,* 70 (1995) 173-190.

[Huegler1997] P.A. Huegler, F.J. Vasko, "A performance comparison of heuristics for the total weighted tardiness problem." *Computers & Industrial Engineering,* 32 (1997) 753-767.

[Kahlbacher1993] H.G. Kahlbacher, "Scheduling with monotonous earliness and tardiness penalties." *European Journal of Operational Research,* 64 (1993) 258-277.

[Rinnooy Kan1975] A.H.G. Rinnooy Kan, B.J. Lageweg, J.K. Lenstra, "Minimizing total costs in one machine schedulling. " *Operation Research,* 23 (1975) 908:927.

[Kanet2007] J.J. Kanet, "New precedence theorems for one-machine weighted tardiness." *Mathematics of Operations Research,* 32(2007) 579-588.

[Karakostas2009] G. Karakostas, S.G. Kolliopoulos, J. Wang, "An FPTAS for the minimum total weighted tardiness problem with a fixed number of distinct due dates." Accepted by *The 15th International Computing and Combinatorics conference (COCOON'2009 Niagara Falls)* .

[Kellerer2006] H. Kellerer and V.A. Strusevich, "A fully polynomial approximation scheme for the single machine weighted total tardiness problem with a common due date." *Theoretical Computer Science,* 369 (2006) 230-238.

[Kolliopoulos2006] S.G. Kolliopoulos and G. Steiner, "Approximation algorithms for minimizing the total weighted tardiness on a single machine." *Theoretical Computer Science,* 355 (3) (2006) 261-273.

[Lawler1977] E.L. Lawler, "A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness." *Annals of Discrete Mathematics,* 1, (1977) 331-342.

[Lawler1982] E.L. Lawler, "A fully polynomial approximation scheme for the total tardiness problem." *Operations Research Letters,* 1, (1982)207-208.

[Leung2004] J.Y.-T. Leung, "Handbook of Scheduling: Algorithms, Models and Performance Analysis." Chapman & Hall/CRC, 2004.

[Liu2003] N. Liu, M.A. Abdelrahman, S. Ramaswamy, "A genetic algorithm for the single machine total weighted tardiness problem." *Porceedings of the 35th Southeastern Symposium on System Theory, 2003,* 16-18 (2003) 34-38.

[Madureira1999] A.M. Madureira, "Meta-heuristics for the single-machine scheduling total weighted tardiness problem." *Proceedings of the 1999 IEEE International Symposium on Assembly and Task Planning, Porto, Portugal,* (1999) 405-410.

[Mathirajan2006] M. Mathirajan, A.I. Sivakumar, "Minimizing total weighted tardiness on heterogeneous batch processing machines with incompatible job families." *The International Journal of Advanced Manufacturing Technology,* 28 (2006) 1038-1047.

[Oguz1994] C. Oguz, C. Dincer, "Single machine earliness-tardiness scheduling problems using the equal-slack rule." *Journal of the Operational Research Society,* 45 (1994) 589-594.

[Osman1996] I.H. Osman, G. Laporte, "Metaheuristics: a bibliography." *Annals of Operations Research,* 63 (1996) 513-623.

[Picard1978] J.C. Picard, M. Queyranne, "The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling." *Operations Research,* 26 (1)(1978)86-110.

[Pinedo1995] M.L. Pinedo, "Scheduling: Theory, Algorithms, and Systems." Prentice-Hall, Upper Saddle River, NJ, 1995.

[Potts1985] C.N. Potts and L.N. Van Wassenhove, "A branch and bound algorithm for the total weighted tardiness problem. " *Operations Research,* 33 (1985) 363-377.

[Potts1991] C.N. Potts and L.N. Van Wassenhove, "Single machine tardiness sequencing heuristics." *IIE Transactions,* 23 (1991) 346-354.

[Qi1998] X.T. Qi, F.S. Tu, "Scheduling a single machine to minimize earliness penalties subject to the SLK due-date determination method." *European Journal of Operational Research,* 105 (1998) 502-508.

[Rachamadugu1987] R.M.V. Rachamadugu, "A note on weighted tardiness problem." *Operations Research,* 35 (1987) 450-452.

[Rinnooy Kan1976] A.H.G. Rinnooy Kan, "Machine sequencing problem: classification, complexity and computation." Nijhoff, The Hague, 1976.

[Schrage1978] L. Schrage, K.R. Baker, "Dynamic programming solution of sequencing problem with precedence constraints." *Operations Research,* 26 (1978) 444-449.

[Sen2003] T. Sen, J.M. Sulek, P. Dileepan, "Static scheduling research to minimize weighted and unweighted tardiness: a state-of-the-art survey." *International Journal of Production Economics,* 83 (2003) 1-12.

[Shwimer1972] J. Shwimer, "On the n-job, one-machine, sequence-independent scheduling problem with tardiness penalties: a branch-bound solution," *Management Science,* 18 (6) (1972) B-301-B-313.

[Szwar1993] W. Szwarc, J.J. Liu, "Weighted tardiness single machine scheduling with proportional weights." *Management Science,* 39 (1993) 626-632.

[Vazirani2001] V.V. Vazirani, "Approximation Algorithms." Springer Verlag, Berlin, 2001.

[Yano1991] C.A. Yano, Y.D. Kim, "Algorithms for a class of single-machine weighted tardiness and earliness problems. " *European Journal of Operational Research,* 52 (1991) 167-178.

[Yuan1992] J. Yuan, "The NP-hardness of the single machine common due date weighted tardiness problem." *Systems Science and Mathematical Sciences,* 5 (1992) 328-333.