An FPTAS for the minimum total weighted tardiness problem with a fixed number of distinct due dates

George Karakostas * Stavros G. Kolliopoulos † Jing Wang ‡

February 3, 2009

Abstract

Given a sequencing of jobs on a single machine, each one with a weight, processing time, and a due date, the tardiness of a job is the time needed for its completion beyond its due date. We present an FPTAS for the basic scheduling problem of minimizing the total weighted tardiness when the number of distinct due dates is fixed. Previously, an FPTAS was known only for the case where all jobs have a common due date.

1 Introduction

The minimum total weighted tardiness problem for a single machine is defined as follows. We are given n jobs, each with a weight $w_j > 0$, processing time p_j, and due date d_j. When these jobs are sequenced on a single machine, each job j will have a completion time C_j. The tardiness T_j of job j is defined as follows

$$T_j := \begin{cases} 0, & \text{if } j \text{ is } \textit{early}, \text{ i.e., } C_j \leq d_j \\ C_j - d_j, & \text{if } j \text{ is } \textit{tardy}, \text{ i.e., } C_j > d_j. \end{cases}$$

The objective is to minimize the total weighted tardiness, i.e., we look for a schedule that minimizes $\sum_j w_j T_j$.

The problem is very basic in scheduling (see surveys [1, 10] and the references in [4, 5]) and is known to be NP-hard [8] even in the case of unit weights [3]. Despite the attention it has received, frustratingly little is known on it approximability. The best known approximation algorithm has a performance guarantee of $n-1$ [2]. For the unit weight case, Lawler gave early on a fully polynomial-time approximation scheme (FPTAS) [7], which is a modification of his pseudopolynomial dynamic programming algorithm in [6].

For general weight values, the problem remains NP-hard even when all jobs have a common due date [11]. Kolliopoulos and Steiner [5] gave a pseudopolynomial dynamic programming algorithm for the case of a fixed number of distinct due dates. Using essentially Lawler's rounding scheme from [7], they obtained an FPTAS only for the case of polynomially bounded weights. Kellerer and Strusevich [4] gave an FPTAS for general weights in the case where all jobs have a common due date. The existence however of an FPTAS for the case of general weights and a fixed number of

*Dept. of Computing & Software, and School of Computational Engineering & Science, McMaster University, Hamilton, ON, Canada. E-mail: karakos@mcmaster.ca. Research supported by an NSERC Discovery grant.

†Dept. of Informatics and Telecommunications, National and Kapodistrian University of Athens, Athens 157 84, Greece. URL: wwww.di.uoa.gr/~sgk

‡School of Computational Engineering & Science, McMaster University, Hamilton, ON, Canada. E-mail: wang257@mcmaster.ca. Research supported by an NSERC Discovery grant.

distinct due dates has remained open. We note that for a general number of distinct due dates the problem becomes strongly NP-hard [6].

In this work, we settle the case of a fixed number of distinct due dates by giving an FPTAS. We design first a pseudopolynomial algorithm and then apply the rounding scheme of [4] to obtain the desired approximation scheme. We exploit two crucial properties of the algorithms in [4]. The first is that the optimal choice is feasible at every job placement the FPTAS performs (cf. Lemma 10). This step-by-step mimicking of the optimal chain of computation is crucial for bounding the approximation error. Of course, the schedule we output may be suboptimal due to our approximate ("rounded") estimation of tardiness. The second property is that the rounding scheme of [4] produces values which correspond to actual schedules; therefore by rounding up the processing time of tardy jobs with due date d, one rounds down the processing time of early jobs with the same due date *by the same amount.* Since the total time needed for these jobs remains the same, this means that there is empty space that allows our algorithm to push back the extra tardy processing time towards the past. This need for *preemption,* i.e., allowing the processing of a job to be interrupted and later restarted, did not arise in [4] where the extra tardy processing time past the common due date D could always be accommodated in the time interval $[D, \infty)$.

In addition to these basic facts, we need a number of other new ideas, some of which we outline next. Our algorithm works in two stages. First, via dynamic programming it computes 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. This is what we call an *abstract schedule.* In the second stage, a greedy procedure allocates the actual job lengths, possibly also with preemption. As in previous algorithms, the jobs that straddle a due date in a schedule, the so-called *straddlers,* play an important role. We observe that only the placement of the tardy straddlers is critical. The time intervals, called *superintervals,* between consecutive tardy straddlers, form the basic time unit on our time horizon. The scheduling of a job j as early can then be localized within only one of these superintervals, depending on the actual d_j value (cf. The Bracketing Lemma 3). This helps to shrink the state space of the dynamic program.

It is well-known that the preemptive and non-preemptive optima coincide when minimizing tardiness on a single machine [9]. This powerful fact has found only limited use in approximation algorithms so far, for example through the preemptive scheduling of early jobs in [5]. We take the opposite view from [5] and insist on the non-preemptive scheduling of early jobs. Moreover, all early jobs are packed explicitly in the abstract schedule. This is necessary since early jobs are particularly difficult to handle: enumerating their total length is prohibitive computationally and distorting their placement even by a tiny amount might result in a severely suboptimal schedule. We allow instead preemptive scheduling of the tardy jobs. As explained above, preemption will allow us to flexibly push back the extra tardy processing time, introduced by the rounding, towards the past. Following this idea to its natural conclusion, we allow even straddlers to be preempted. In the final schedule, it could be that only the completion time of a tardy job happens in the interval in which it was originally assigned by the dynamic program, while all the processing happens earlier. The algebraic device we introduce that allows the abstract schedule to keep some of the jobs "floating", without pinning down anything but their completion time, is the *potential* empty space within a prefix of a schedule (cf. Eq. (3) below). To ensure that preemptions can be implemented into *actual* empty space is perhaps the largest technical difficulty in our proof.

The approximability of total weighted tardiness problem with an arbitrary number of distinct due dates remains as the main open problem.

2 Structural properties of an optimal schedule

We are given n jobs $j = 1, \ldots, n$, each with its own processing time p_j and weight w_j and a due date from a set of K possible distinct due dates $\{d_1, d_2, \ldots, d_K\}$, where K will be assumed to be a constant for the rest of this paper. For convenience, we are also going to define the artificial due date $d_0 = 0$. The due dates partition the time horizon into $K+1$ intervals $I_l = [d_{l-1}, d_l)$ for $l = 1, \ldots, K$, and $I_{K+1} = [d_K, \infty)$. We partition the jobs into K classes $C_1, C_2, \ldots, C_K$ according to their due dates.

A crucial concept for the algorithms we describe is the grouping of intervals I_l in the following manner: for any i_u, i_{u+1}, intervals $I_{i_u+1}, I_{i_u+2}, \ldots, I_{i_{u+1}}$ are grouped into a *superinterval* $G_{i_u i_{u+1}} = I_{i_u+1} \cup I_{i_u+2} \cup \ldots \cup I_{i_{u+1}} = [d_{i_u}, d_{i_{u+1}})$, if straddlers S_{i_u} and $S_{i_{u+1}}$ are consecutive tardy straddlers, i.e., there is no other tardy straddler in between due dates $d_{i_u}, d_{i_{u+1}}$. Note that it may be the case that $i_{u+1} = i_u + 1$, i.e., $G_{i_u i_{u+1}} \equiv I_{i_u+1}$ if both S_{i_u}, S_{i_u+1} are tardy. Also, since straddler S_K is tardy, the last superinterval is $G_{K,K+1} = I_{K+1}$.

In any schedule of the n jobs, a job that finishes before or on its due date will be an *early* job, otherwise it will be *tardy*. We also call any job that starts before or on a due date but finishes after it a *straddler*. It is well-known [9] that the optimal values of the preemptive and the non-preemptive version of the problem are the same. Therefore we can assume that the optimal schedule is a non-preemptive one. In it the straddlers will appear as contiguous blocks, crossing one or more due dates. For easiness of exposition, we will assume that there is an optimal schedule with distinct straddlers for every due date, i.e., there are K distinct straddlers $S_1, \ldots, S_K$ corresponding to due dates $d_1, \ldots, d_K$. After the description of the algorithms, it should be clear how to modify them in order to deal with the special case of some straddlers crossing more than one due dates. For convenience, let also S_0 be an artificial tardy straddler for d_0 with $w_{S_0} = p_{S_0} = 0$. In any optimal schedule, the machine has clearly no idle time. Hence, wlog, due dates that are greater than $\sum_j p_j$, can be set to ∞. Accordingly, we can assume that there is a straddler for every due date.

Tardy straddlers are going to be of particular interest to what our algorithms do. We will assume that we have guessed the number $M \leq K$ of tardy straddlers and these tardy straddlers $S_{i_1}, \ldots, S_{i_M}$ of the optimal schedule (also $S_{i_0} = S_0$). By guessing, we mean the exhaustive enumeration of all combinations of jobs with due dates (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 constant. Let $m = n - M$ be the number of the remaining jobs, which are ordered according to their weighted shortest processing times (WSPT), i.e., $\frac{p_1}{w_1} \leq \frac{p_2}{w_2} \leq \ldots \leq \frac{p_m}{w_m}$. With some abuse of terminology, we will call these jobs *non-straddling*, although some of them are the early straddlers. We will also assume that we have guessed a bound Z^{ub} such that for the optimal value OPT we have $Z^{ub}/2 \leq OPT \leq Z^{ub}$.[1]

It should be obvious that, in any interval I_l, the tardy jobs in that interval are processed before the early ones. It is also well-known (e.g., see Lemma 2.1 in [5]) that the tardy jobs must be processed in WSPT order. With respect to a given partial schedule, we define the following quantities, which are a going to be important throughout this work:

- $y_k^{(i-1)t},\ 1 \leq t < i \leq K+1,\ 1 \leq k \leq m$: the total processing time of those (tardy) jobs among the first k (in WSPT order) jobs, that belong to class C_t and are processed in I_i. Also define $y_k^{0t} = 0$ for all t.
- $W_k^{(i-1)t},\ 1 \leq t < i \leq K+1,\ 1 \leq k \leq m$: the total weight of the jobs in the previous item.

[1] This can be done by running the algorithm with $Z^{ub} = 2^x$, for all $x = 0, 1, \ldots, U$, with U being a trivial upper bound of OPT, e.g. $U = \log(n^2 w_{max} p_{max}) = O(\log n + \log w_{max} + \log p_{max})$.

- $A_k^t,\ 1 \le t \le K,\ 1 \le k \le m$: the total processing time of the class C_t jobs among the first k jobs. Notice that these quantities can be calculated in advance.
- $e_k^{it},\ 1 \le i \le t \le K,\ 1 \le k \le m$: the total processing time of those (early) jobs among the first k (in WSPT order) jobs, that belong to class C_t and are in I_i.

The following lemmas are important properties of an optimal schedule:

Lemma 1 *In the optimal schedule and for any $1 \le i \le K$, if S_i is tardy, then for any $1 \le l \le i$ and any $i+1 \le u \le K$, we have $e_k^{lu} = 0$.*

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

Lemma 2 *In the optimal schedule and for any $2 \le i \le K$, if S_{i-1} is early, then $y_k^{(i-1)u} = 0$ for any $1 \le u \le i-1$, i.e., there are no tardy jobs in I_i.*

Proof: Suppose there exists $2 \le i \le K$ such that S_{i-1} is early, while $y_k^{(i-1)u} > 0$. Then there are some C_u jobs ($1 \le u \le i-1$) which are tardy in I_i. Then exchanging part of S_{i-1} with some or part of these C_u jobs will reduce their total tardiness, and S_{i-1} is still early. This is a contradiction of optimality. □

Lemma 2 implies that the only non-zero y's are the ones that correspond to the first interval of each superinterval. Therefore, from now on we will use only the values $y_k^{i_u t},\ 1 \le u \le M,\ 1 \le t \le i_u,\ 1 \le k \le m$. Lemmas 1 and 2 imply that for every $1 \le k \le m$ and for every $1 \le t \le K$ s.t. $i_{s-1} < t \le i_s$ for some $1 \le s \le M$ we have

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

A direct consequence of Lemma 1 and the definition of a superinterval is the following.

Lemma 3 (Bracketing Lemma for early jobs) *Let $u \le M$. In an optimal schedule only jobs from classes C_t, with $i_{u-1} < t \le i_u$ can be assigned as early in the superinterval $G_{i_{u-1} i_u}$.*

3 A dynamic programming algorithm to find an abstract schedule

An *abstract schedule* is an assignment of the the m non-straddling jobs to superintervals so that (i) early jobs are feasibly and non-preemptively packed within their assigned superinterval (ii) there is enough empty space so that tardy jobs that complete in their assigned superinterval can be preemptively packed and (iii) there is enough empty space so that the M tardy straddlers can be preemptively packed. An *abstract k-schedule, $k \le m$*, is an abstract schedule for the first k non-straddling jobs. In this section we describe a pseudopolynomial dynamic programming algorithm (DP) that computes a suitable abstract schedule. In the next section we show how to pin down the actual processing of the tardy jobs and the straddlers, so that the abstract schedule is converted to an actual schedule of the n jobs with minimum total tardiness.

The DP algorithm "guesses" the M tardy straddlers. Extending the dynamic programming of [4], the states of DP store the following values for a (partial) schedule of the k first (in WSPT order) of the m non-straddling jobs[2]:

$$\left(k, Z_k, y_k^{i_1 1}, W_k^{i_1 1}, y_k^{i_2 1}, W_k^{i_2 1}, \cdots, y_k^{i_M 1}, W_k^{i_M 1}, y_k^{i_1 2}, W_k^{i_1 2}, \cdots, y_k^{i_M K}, W_k^{i_M K}\right), \tag{2}$$

where Z_k is the total weighted tardiness of the k scheduled jobs. Note that some of the $y_k^{i_u j}, W_k^{i_u j}$ in (2) may not exist, if $i_u < j$. As in [4], the weight values $W_k^{i_u j}$ will be needed when the tardy straddlers will be re-inserted at the end.

The initial state will be $(0, 0, \ldots, 0)$. A *state-to-state transition* from state (2) corresponds to the insertion of the $(k+1)$-th job in 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 superinterval, 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 superinterval, 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 superinterval $G_{i_{u-1} i_u}$ chosen for it by the transition determines whether this job is early or tardy: if $1 \le t \le i_{u-1}$ then the job is tardy, otherwise it is early.

In order to be able to check the feasibility of the transitions, we would like to be able to calculate the empty space in every superinterval from the information stored in states (2). Unfortunately, this is not possible, because essentially there are many possibilities for the placement of early jobs that yield the same state and keeping track of all these possibilities would blow up the state space. As a result of this limited information, some of the space that looks empty will be actually needed to accommodate preempted parts of tardy jobs from later superintervals. Nevertheless, we can calculate the potential empty space for *prefixes* of the schedule that start from time $t = 0$. The processing time for a tardy job is just slated for the prefix that ends at its assigned completion time by the first (dynamic programming) stage of the algorithm, without pinning down its exact placement. This placement is fixed only during the second stage of the algorithm. We introduce the following set of prefix values, which can be calculated given a state (2):

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

Given $1 \le l \le K$, let s be such that $i_{s-1} < l \le i_s$. Then L_k^{0l} can be computed from the information at hand as follows:

$$\begin{aligned} L_k^{0l} &= d_l - \Big(\sum_{j=1}^{s-1} \sum_{q=i_{j-1}+1}^{i_j} \sum_{h=q}^{i_j} e^{qh} + \sum_{q=i_{s-1}+1}^{l} \sum_{h=q}^{l} e^{qh}\Big) - \Big(\sum_{j=1}^{s-1} \sum_{h=1}^{i_j} y^{i_j h}\Big) \\ &= d_l - \Big(\sum_{i=1}^{l} A_k^i - \sum_{j=1}^{s-1} \sum_{h=1}^{i_j} y^{i_j h} - \sum_{j=s}^{M} \sum_{h=1}^{l} y^{i_j h}\Big) - \Big(\sum_{j=1}^{s-1} \sum_{h=1}^{i_j} y^{i_j h}\Big) \\ &= d_l - \sum_{i=1}^{l} A_k^i + \sum_{j=s}^{M} \sum_{h=1}^{l} y^{i_j h} \end{aligned} \tag{3}$$

[2]Recall that we are looking for schedules that do not include the tardy straddlers, yet they have enough empty space to accommodate the re-insertion of these straddlers in their correct position. Moreover, in every interval I_l, the tardy jobs of that interval (if they exist) appear as a block starting at d_{l-1}, followed immediately by the block of early jobs in this interval.

Recall that there are M tardy straddlers $\{S_{i_u}\}_{u=1}^{M}$ overall. We assume that the $(k+1)$-th job J_{k+1} belongs to class C_t, and that we want to schedule it in superinterval $G_{i_{u-1}i_u}$. Note that Lemma 3 implies that, to even consider such a placement, $t \leq i_u$ must hold. The three feasibility conditions that must be satisfied by a DP transition from state (2) follow. From equation (3), given the state information, all three can be effectively checked.

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

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

1b. If 1a doesn't hold, check whether $L_k^{0l} \geq p_{k+1}$ holds $\forall l$ s.t. $i_{u-1} < l \leq i_u$.

1c. Check whether $L_k^{0i_j} \geq p_{k+1}$ holds $\forall j$ s.t. $u < j \leq M$.

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

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

2b. If 2a doesn't hold, check the following according to which case applies:

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

2b.2. $\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} > L_k^{0i_{u-1}}$: Check whether $L_k^{0l} \geq p_{k+1}$ holds $\forall l$, s.t. $t \leq l \leq i_u$.

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

Condition (3): Check whether $L_{k+1}^{0j} \geq \sum_{h=1}^{u-1} p_{i_h}$ 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 (Lemma 8). Conditions (1a) (and (2a)) are satisfied when there is enough space to fit J_{k+1} as tardy (or early) in a *non-preemptive* schedule. Since we will prove (Lemma 6) that Conditions (2b), (2c) are enough to guarantee (with a some shuffling around) that early jobs can always be inserted non-preemptively in a preemptive schedule, and Lemma 7 will show that even if Condition (1a) is not satisfied, we are able to insert tardy jobs preemptively in a preemptive schedule if Conditions (1b), (1c) hold, Conditions (1a),(2a) are redundant if we are looking for a preemptive schedule. But we will use the fact that Conditions (1a),(2a),(3) are enough for the construction of an optimal DP algorithm which produces an optimal *non-preemptive* schedule in the analysis of our FPTAS (Sections 4, 5).

There is a more concise way of expressing Condition (2), as shown in the following

Lemma 4 *Condition (2b) can be replaced by the following:*
2b. 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: We give the proof of deriving 2b.2 (the rest of the proof is obvious): if $\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} > L_k^{0i_{u-1}}$, 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 (3), we know that $L_k^{0l} - L_k^{0i_{u-1}} = d_l - d_{i_{u-1}} - (\sum_{q=i_{u-1}+1}^{l} \sum_{v=q}^{l} e_k^{qv} + \sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v})$ for all $t \leq l \leq i_u$. An interpretation of these inequalities is that for all the class l jobs which are early in the superinterval $G_{i_{u-1}i_u}$ 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 superinterval $G_{i_{u-1}i_u}$ is computed as follows:

- J_{k+1} is **early**: Set $Z_{k+1} = Z_k$, $y_{k+1}^{i_u j} = y_k^{i_u j}$, $W_{k+1}^{i_u j} = W_k^{i_u j}$ for all $1 \leq u \leq M$, $1 \leq j \leq i_u$.
- J_{k+1} is **tardy**: Set $Z_{k+1} = Z_k + w_{k+1}(\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} + p_{k+1} + d_{i_{u-1}} - d_t)$, $y_{k+1}^{i_{u-1}t} = y_k^{i_{u-1}t} + p_{k+1}$, $W_{k+1}^{i_{u-1}t} = W_k^{i_{u-1}t} + w_{k+1}$. 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.

We need to show that the assignment of jobs to the superintervals meets the definition of the abstract schedule. First we elucidate the relation of the L values with the actual empty space.

Lemma 5 *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 superintervals.*

Proof: Note that these tardy jobs must each be scheduled so that they complete in their respective superinterval. Their processing can take place anywhere before their completion time. For a superinterval $G_{i_{j-1}i_j}$, define $L_k^{i_{(j-1)}i_j} := L_k^{0i_j} - L_k^{0i_{(j-1)}}$. By Lemma 3 this quantity equals 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}. Clearly $L_k^{0i_u} = \sum_{j=1}^{u} L_k^{(i_{j-1})i_j}$. Each of the terms in the sum can be negative or nonnegative. A negative term corresponds to a superinterval with an excess portion of tardy jobs which needs to be moved (preempted) towards the past. A nonnegative term corresponds to a superinterval with an excess of space which can be used to accommodate preempted parts of jobs that complete in future superintervals. Therefore, if $L_k^{0i_h} \geq 0$, $\forall h$ s.t. $1 \leq h \leq j$, the sum $L_k^{0i_j} \geq 0$, is the net empty space available 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. □

We establish that the early jobs are feasibly packed.

Lemma 6 *Assume state (2) corresponds to an abstract k-schedule. Conditions (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: If Condition (2a) holds, there is at least p_{k+1} empty space in the superinterval $G_{i_{u-1}i_u}$ although (i) it may not be contiguous (ii) it may not occur in its entirety before d_t (iii) part of it may be earmarked to accommodate preemptions from tardy jobs assigned after d_{i_u}. If Condition (2b) holds, one has in addition to move parts of tardy jobs from $G_{i_{u-1}i_u}$ towards the past in order to create the empty space of (2a). If neither of them holds, it is impossible to pack J_{k+1} as early within this superinterval. We establish that assigning J_{k+1} under Conditions (2a) or (2b) has no ill effect on the first k jobs. Then we consider how the possibly fragmented empty space can be used to feasibly pack J_{k+1}.

After assigning J_{k+1} to $G_{i_{u-1}i_u}$, $L_{k+1}^{0i_j} = L_{k+1}^{0i_j}$, $\forall j$, s.t. $1 \leq j \leq i_{u-1}$. By Lemma 5, the feasible assignment of jobs to intervals before $d_{i_{u-1}}$ is not affected. Space for straddlers is preserved because

of Condition (3). Early jobs assigned after d_{i_u} are not affected either. We only have to worry about tardy jobs assigned after $d_{i_{u-1}}$ and early jobs in the superinterval $G_{i_{u-1}i_u}$. The former can afford to lose some of their coveted space because of Lemma 5 and Conditions (2a) (or (2b)) and (2c). The latter are packed according to the scheme that follows. Since our reasoning applies regardless of whether Condition (2a) or (2b) holds let $\bar{L}_k^{i_{u-1}l}$ denote $L_k^{0l} - L_k^{0i_{u-1}}$ in the former case and $d_l - d_{i_{u-1}} - (\sum_{q=i_{u-1}+1}^{l} \sum_{v=q}^{l} e_k^{qv} + \max\{\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} - L_k^{0i_{u-1}}, 0\})$ in the latter (from Lemma 4).

Recall that $i_{u-1} < t \leq i_u$. We have $\bar{L}_k^{i_{u-1}i_u} \geq p_{k+1}$, and that $\bar{L}_k^{i_{u-1}(i_u-1)}$ is the space that is either empty or contains (parts of) jobs from class C_{i_u} (due to Lemma 1) in $[d_{i_{u-1}}, d_{i_u-1})$. We can greedily "push" all the latter jobs as close to d_{i_u} as possible using the empty space closest to d_{i_u}. After we are done, the empty space between $d_{i_{u-1}}$ and d_{i_u-1} must be at least p_{k+1}: If all C_{i_u} jobs fit in the empty space of I_{i_u}, then $\bar{L}_k^{i_{u-1}(i_u-1)}$ represents actual empty space and by Condition (2) $\bar{L}_k^{i_{u-1}(i_u-1)} \geq p_{k+1}$; otherwise, there must be no empty space left in I_{i_u}, which means that the whole empty space in $G_{i_{u-1}i_u}$, which we know to be at least p_{k+1}, is concentrated in $[d_{i_{u-1}}, d_{i_u-1})$. Continuing to successively push jobs of classes $C_{i_u-1}, C_{i_u-2}, \ldots, C_{t+1}$ as close as possible to due dates $d_{i_u-1}, d_{i_u-2}, \ldots, d_{t+1}$ respectively, at the end we will have at least p_{k+1} units of empty space in $[d_{i_{u-1}}, d_t]$, and in this empty space we can insert (early) job J_{k+1}, without disturbing the previous k jobs just as the statement of the lemma specifies. □

It remains to argue about the packing of tardy jobs.

Lemma 7 *Assume state (2) 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 superinterval $G_{i_{u-1}i_u}$, so that we obtain an abstract $(k+1)$-schedule.*

Proof: The proof is very similar to the proof of Lemma 6. We argue first that Conditions (1a) (or (1b)), (1c) and (3) do not affect the assignment of the first k jobs. If Condition (1a) holds, there is at least p_{k+1} empty space in the superinterval $G_{i_{u-1}i_u}$ although (i) it may not be contiguous (ii) part of it may be earmarked to accommodate preemptions from tardy jobs assigned after d_{i_u}. Condition (1b) corresponds to the assignment of job J_{k+1} as "floating" in the prefix $[d_0, d_{i_u})$. In both cases, we may need to shift the early jobs of the superinterval as in the previous proof. □

4 Producing an optimal schedule

The abstract schedule produced so far by the dynamic programming algorithm has placed the early jobs in their superintervals non-preemptively and as close to their due date as possible (as shown by Lemma 6). It has also placed the completion times of the tardy jobs in their superintervals[3]. But we have not specified how the (preempted) tardy jobs are arranged, since Condition (1) only ensures that there is enough empty space to fit each tardy job, possibly broken in pieces. Now we describe the procedure that allocates the tardy jobs on the time horizon:

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

2. 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

[3]In fact, we know the specific *interval* of each completion time, since only the first interval of every superinterval can be used for the completion of tardy jobs.

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 doesn't matter). When we run out of either empty space or preempted pieces, we move to the next $u := u - 1$.

We 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.

The placement of the tardy straddlers will complete the schedule the algorithm will output. The following lemma shows how we will place the straddlers preemptively so that two properties are maintained: (a) straddler S_{i_u} completes at or after d_{i_u} and before $d_{i_{u+1}}$, for all $u = 1, 2, \ldots, M-1$, and (b) the prefix of the schedule that contains all straddlers' processing time is *contiguous*, i.e., there are no 'holes' of empty space in it. We will need property (b) in the calculation of the total tardiness of the final schedule below and in our FPTAS. We emphasize that (b) 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 $d_{j_1}, \ldots, d_{i_M}$ respectively.

Lemma 8 *The placement of the tardy straddlers can be done so that properties (a),(b) above are maintained.*

Proof: First we note that the quantity L_m^{0K} is the actual empty space in $[d_0, d_K)$, and since Condition (3) is true, for $u = M, j = i_M$ we have $L_m^{0i_M} \geq \sum_{h=1}^{M-1} p_{S_{i_h}}$, i.e., we have enough actual empty space in $[d_0, d_K)$ for all tardy straddlers other than S_{i_M}.

If $L_m^{0i_M} - \sum_{h=1}^{M-1} p_{S_{i_h}} \leq p_{S_M}$, 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_M}, and fit the rest $p_{S_M} - (L_m^{0i_M} - \sum_{h=1}^{M-1} p_{S_{i_h}})$ units right after d_{i_M} (shifting the jobs in $I_{K,K+1}$ towards the future by an equal amount of units). Otherwise (i.e., $L_m^{0i_M} - \sum_{h=1}^{M-1} p_{S_{i_h}} > p_{S_M}$), set the completion time of S_{i_M} at d_{i_M}, leave $L_m^{0i_M} - \sum_{h=1}^{M} p_{S_{i_h}}$ units of empty space closest to d_{i_M} empty, and fit S_{i_M} right before this empty space.

After the placement of S_{i_M} as described, we are left with exactly $\sum_{h=1}^{M-1} p_{S_{i_h}}$ units of actual empty space before S_{i_M}, and in this space we fit *exactly* $S_{i_1}, \ldots, S_{i_{M-1}}$ in this order. As a result, property (b) is fulfilled. To prove property (a), we use arguments similar to the shifting scheme in Lemma 6. We look at each due date d_{i_u} for $u = M-1, M-2, \ldots, 1$: For $u = M-1$, note that $L_m^{0,i_{M-1}+1}$ still contains in it at least $\sum_{h=1}^{M-1} p_{S_{i_h}}$ units of empty space, because of Lemma 6, Condition (3) that states $L_m^{0,i_{M-1}+1} \geq \sum_{h=1}^{M-1} p_{S_{i_h}}$, and the fact that there are no preempted pieces counted in $L_m^{0,i_{M-1}+1}$ yet. Therefore we know that $S_{i_1}, \ldots, S_{i_{M-1}}$ will be fitted in $[d_0, d_{i_{M-1}+1})$, which implies that $S_{i_{M-1}}$ is placed correctly. Now we consider two cases (as in Lemma 6):

- if interval $I_{i_{M-1},i_{M-1}+1}$ contains any empty space, 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, $L_m^{0,i_{M-1}}$ in Condition (3) inequality $L_m^{0,i_{M-1}} \geq \sum_{h=1}^{M-2} p_{S_{i_h}}$ doesn't contain any preempted parts, and because of the way early jobs were pushed in Lemma 6, we can conclude that $[d_0, d_{i_{M-1}})$ contains at least $\sum_{h=1}^{M-2} p_{S_{i_h}}$ units of empty space.

- if interval $I_{i_{M-1},i_{M-1}+1}$ contains no empty space, all the empty space of $L_m^{0,i_{M-1}+1}$ (which is at least $\sum_{h=1}^{M-1} p_{S_{i_h}} \geq \sum_{h=1}^{M-2} p_{S_{i_h}}$) actually is in $[d_0, d_{i_{M-1}})$.

Both cases imply that $[d_0, d_{i_{M-1}})$ contains at least $\sum_{h=1}^{M-2} p_{S_{i_h}}$ units of empty space, and we can continue in the same manner as before to show that $S_{i_{M-2}}$ is correctly placed, then $S_{i_{M-3}}$, etc. □

Given that Z^{ub} is large enough, the dynamic programming will ultimately produce a set of states with their first coordinate equal to m, i.e., states that correspond to partial schedules of all m non-straddling jobs. Since these states satisfy Condition (3), Lemma 8 implies that we can re-insert the straddlers at their correct position without affecting the earliness of the early or the placement in intervals of the tardy non-straddling jobs, thus creating a number of candidate full schedules. Let $\{T_{i_u}\}_{u=1}^{M}$ be the tardiness of the M tardy straddlers. Also, note that due to property (b) in Lemma 8,

$$x_{i_u} := \max\{0, \sum_{l=1}^{u} p_{S_{i_l}} - L_m^{0i_u}\} \tag{4}$$

is the part of S_{i_u} beyond due date d_{i_u}. Then, if $S_{i_u} \in C_t$ (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 a candidate schedule is

$$Z = Z_m + \sum_{u=1}^{M} w_{i_u} T_{i_u} + \sum_{u=1}^{M} (\sum_{l=1}^{i_u} W_m^{i_u l}) x_{i_u}. \tag{5}$$

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. It should be obvious how to extend the description of the algorithm above to include the case of a straddler being the same for more than one (consecutive) due dates. The following is also fairly easy to prove:

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

Proof: Take any optimal non-preemptive schedule (which we already know that exists) and remove the straddlers. Consider also the sequence of partial schedules that result by removing jobs $\{J_2, J_3, \ldots, J_m\}, \{J_3, \ldots, J_m\}, \ldots, \{J_m\}$ respectively. We will show that these partial schedules can be produced by the algorithm, i.e., Conditions (1)-(3) hold for every placement of a job in the superinterval 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). Conditions (1a) and (2a) (depending on whether the $(k+1)$-th job is tardy or early in its superinterval in the optimal schedule) also hold. For example, for Condition (2a) (the argument is the same for Condition (1a)), assume that job $J_{k+1} \in C_t$ is inserted early in $G_{i_{u-1} i_u}$ and is the first for which Condition (2a) is not true, i.e., it holds that $L_k^{0l} - L_k^{0i_{u-1}} < p_{k+1}$ for some l s.t. $t \leq l \leq i_u$. Then we have $\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1} v} + \sum_{v=i_{u-1}+1}^{l} \sum_{q=i_{u-1}+1}^{v} e_k^{qv} + p_{k+1} > d_l - d_{i_{u-1}}$ (recall that jobs $J_1, J_2, \ldots, J_k$ have been inserted non-preemptively). This means that the space in $[d_{i_{u-1}}, d_l]$ is not enough to fit all tardy jobs and early jobs with due dates $v \leq l$ among $\{J_1, J_2, \ldots, J_k\}$ that have been assigned to $G_{i_{u-1} i_u}$ by the optimal schedule. This contradicts the fact that all these jobs could be fitted there, as the optimal schedule shows. Similarly, we can show that Conditions (1c),(2c) also hold.

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.

□

Note that in the proof of Theorem 1 we didn't need to check Conditions (1b),(2b). If, in addition, we require that the algorithm is non-preemptive, then the proof goes through without checking for Conditions (1c),(2c), since they are satisfied trivially by the optimal non-preemptive schedule. Hence we have the following

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.*

Corollary 1 will be important for the proof of the approximation ratio guarantee below, since we will compare the solution produced by our FPTAS to the optimal schedule of the corollary.

5 The FPTAS

The transformation of the pseudopolynomial algorithm described in Sections 3, 4 into an FPTAS follows closely the FPTAS (Algorithm **Eps**) in [4]. Since the running time of the dynamic programming part dominates the total running time, in what follows we use the term DP to refer to the entire process.

Let $\varepsilon > 0$. Recall that we have guessed Z^{ub} such that $Z^{ub}/2 \leq OPT \leq Z^{ub}$, and let $Z^{lb} := Z^{ub}/2$. Define $\delta = \frac{\varepsilon Z_{lb}}{4m}$. Consider a state

$$\left(k, Z_k^*, y_k^{i_1 1*}, W_k^{i_1 1*}, y_k^{i_2 1*}, W_k^{i_2 1*}, \cdots, y_k^{i_M 1*}, W_k^{i_M 1*}, y_k^{i_1 2*}, W_k^{i_1 2*}, \cdots, y_k^{i_M K*}, W_k^{i_M K*}\right)$$

of the exact dynamic programming. From this state, we will deduce the states

$$\left(k, Z_k, y_k^{i_1 1}, W_k^{i_1 1}, y_k^{i_2 1}, W_k^{i_2 1}, \cdots, y_k^{i_M 1}, W_k^{i_M 1}, y_k^{i_1 2}, W_k^{i_1 2}, \cdots, y_k^{i_M K}, W_k^{i_M K}\right)$$

used by the FPTAS dynamic programming as follows:

We round variable Z_k^* to the next multiple of δ (hence Z_k takes at most $\frac{Z^{ub}}{\delta} = O(\frac{n}{\varepsilon})$ distinct values). For every $1 \leq u \leq M$, we round $W_k^{i_u j*}$ to the nearest power of $(1+\varepsilon/2)^{1/m}$ (hence $W_k^{i_u j}$ takes $O(n \log W)$ values, where W is the total weight of the n jobs). After ordering the non-straddling jobs in WSPT order, let $w_{\pi(1)} > w_{\pi(2)} > \cdots > w_{\pi(N)}$ be the $N \leq m$ distinct weight values of the m non-straddlers in decreasing order.

The rounding of $y_k^{i_u j*}$, $1 \leq u \leq M$ is more complicated. Define a division of time interval $[0, \frac{Z^{ub}}{w_{\pi(N)}}]$ into subintervals $\{H_{i'} := [\frac{Z^{ub}}{w_{\pi(i'-1)}}, \frac{Z^{ub}}{w_{\pi(i')}}]\}_{i'=1}^{N}$. In turn, divide each $H_{i'}$ into subintervals $\{\hat{H}_{i'j'}(i)\}_{j'=1}^{x_i^{i'}}$ of length $\delta_i = \frac{\delta}{i w_{\pi(i')}}$ for all $1 \leq k \leq m$ and $1 \leq i \leq K$, where $x_i^{i'} = \lceil \frac{\frac{Z^{ub}}{w_{\pi(i')}} - \frac{Z^{ub}}{w_{\pi(i'-1)}}}{\delta_i} \rceil$ is the number of such subintervals (note that the length of the last subinterval may be less than δ_i). For each state $(k, Z_k, y_k^{i_1 1}, W_k^{i_1 1}, \cdots, y_k^{i_M K}, W_k^{i_M K})$, the dynamic program applies its $O(K)$ transitions to generate new states $(k+1, Z_{k+1}, y_{k+1}^{i_1 1}, W_{k+1}^{i_1 1}, \cdots, y_{k+1}^{i_M K}, W_{k+1}^{i_M K})$. 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: we 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}^{ij}, i.e., $(k+1, Z_{k+1}, y_{k+1}^{i_1 1}, W_{k+1}^{i_1 1}, \cdots, y_{k+1}^{i_u j^{max}}, \cdots y_{k+1}^{i_M K}, W_{k+1}^{i_M K})$ and $(k+1, Z_{k+1}, y_{k+1}^{i_1 1}, W_{k+1}^{i_1 1}, \cdots, y_{k+1}^{i_u j^{min}}, \cdots y_{k+1}^{i_M K}, W_{k+1}^{i_M K})$.

Lemma 9 *The algorithm runs in time* $O((\varepsilon^{-1} n \log W \log P)^{\Theta(K^2)})$.

Proof: Assume the worst case $M = K$. For each one of the $K(K+1)/2$ positions $y_{k+1}^{i_u j}$ we have at most

$$\frac{\frac{Z^{ub}}{w_{\pi(1)}}}{\delta_i} + \sum_{i=2}^{N} \frac{(\frac{Z^{ub}}{w_{\pi(i)}} - \frac{Z^{ub}}{w_{\pi(i-1)}})}{\delta_i} = O(\frac{n^2}{\varepsilon})$$

distinct subintervals, or $O((\frac{n^2}{\varepsilon})^{\frac{K(K+1)}{2}})$ combinations of subintervals. When the combination of subintervals is fixed, we have $2^{\frac{K(K+1)}{2}}$ combinations of possible values for the $y_{k+1}^{i_u j}$'s, since there are two choices for each of them. Therefore, for the same values of $Z_{k+1}, W_{k+1}^{i_1 1}, \cdots, W_{k+1}^{i_K K}$, we have $O((\frac{n^2}{\varepsilon})^{\frac{K(K+1)}{2}} 2^{\frac{K(K+1)}{2}}) = O(\varepsilon^{-\frac{K(K+1)}{2}} n^{K(K+1)})$ states. Taking into account the rest of the state values and the initial guessing part (straddlers & Z^{ub}), overall the algorithm runs in $O((\varepsilon^{-1} n \log W \log P)^{\Theta(K^2)})$ time, where W, P are the total weight and total processing time of all jobs. □

We focus on states $(k, Z_k^*, y_k^{i_1 1 *}, W_k^{i_1 1 *}, \cdots, y_k^{i_M K *}, W_k^{i_M K *})$, $k = 0, 1, \ldots, m$ that are the sequence of transitions in the DP of Corollary 1 that produces an optimal non-preemptive schedule. The following lemma shows that despite the rounding used after every transition in our algorithm, there is a sequence of states $(k, Z_k, y_k^{i_1 1}, W_k^{i_1 1}, \cdots, y_k^{i_M K}, W_k^{i_M K})$, $k = 0, 1, \ldots, m$ whose transitions from one state to the next *match exactly* the job placement decisions of the optimal DP step-for-step. The key idea is that when our algorithm overestimates the space needed by tardy jobs (i.e., the y's are rounded *up*), the space needed by the corresponding early jobs is decreased (rounded *down*), since the total space needed remains the same, as (1) shows. The preemption of the tardy jobs allows us to treat the total space taken by the jobs in a class C_t as a unified entity, because the overestimated processing time of tardy jobs in this class can be placed (preempted) in the place of early jobs, whose processing time is reduced by an equal amount. This is the basic motivation behind our introduction of tardy job preemption.

Lemma 10 *For every $k = 1, 2, \ldots, m$, given the identical placement of the first $k-1$ jobs, if a placement of job J_k is feasible for the optimal DP, then the same placement is feasible for our DP.*

Proof: We use induction. Obviously the lemma is true for $k = 1$, since both DPs start from the same initial state $(0, 0, 0, \ldots, 0)$. Assuming that it is true up to the placement of job J_k, i.e., the optimal and our partial schedules have identical placements of jobs $J_1, J_2, \ldots, J_k$ in superintervals, we look at the placement of job J_{k+1}. In what follows, starred quantities refer to the optimal schedule, and non-starred ones to ours. Let $J_{k+1} \in C_t$, and suppose that the optimal placement is in superinterval $G_{i_{u-1} i_u}$. Throughout the proof, we will use the fact that $L_k^{0l} \geq L_k^{0l*}$ $\forall l$, s.t. $1 \leq l \leq K$, due to the identical placement of the first k jobs, Eq. (3), and the fact that the y's are always rounded *up*. We distinguish two cases, according to the optimal placement of J_{k+1}:

Case 1: J_{k+1} is early. Since J_{k+1} is early, $i_{u-1} < t \leq i_u$ and $L_k^{0l*} - L_k^{0 i_{u-1} *} \geq p_{k+1}$ holds, $\forall l$ s.t. $t \leq l \leq i_u$. Therefore we have $L_k^{0l*} \geq L_k^{0l*} - L_k^{0 i_{u-1} *} \geq p_{k+1}$.

In our algorithm, if $L_k^{0l} - L_k^{0 i_{u-1}} \geq p_{k+1}$ holds $\forall l$ s.t. $t \leq l \leq i_u$, Condition (2a) is satisfied. Otherwise, we examine the two cases of Condition (2b):

1. $\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} \le L_k^{0i_{u-1}}$: Since $\sum_{q=i_{u-1}+1}^{l} \sum_{v=q}^{l} e_k^{qv}$ has always been rounded down, we have

$$d_l - d_{i_{u-1}} - (\sum_{q=i_{u-1}+1}^{l} \sum_{v=q}^{l} e_k^{qv}) \ge d_l - d_{i_{u-1}} - (\sum_{q=i_{u-1}+1}^{l} \sum_{v=q}^{l} e_k^{qv*}) \ge L_k^{0l*} - L_k^{0i_{u-1}*} \ge p_{k+1}$$

2. $\sum_{v=1}^{i_{u-1}} y_k^{i_{u-1}v} > L_k^{0i_{u-1}}$: Since $L_k^{0l} \ge L_k^{0l*}$, we have $L_k^{0l} \ge p_{k+1},\ t \le l \le i_u$. Also, since $L_k^{0i_j*} \ge p_{k+1}$ holds, we have $L_k^{0i_j} \ge L_k^{0i_j*} \ge p_{k+1},\ \forall j$, s.t. $u < j \le M$.

Additionally, Condition (2c) is satisfied, because $L_k^{0i_j} \ge L_k^{0i_j*} \ge p_{k+1},\ \forall u < j \le M$. Hence, Condition (2) is satisfied by state $(k, Z_k, y_k^{i_1 1}, W_k^{i_1 1}, \cdots, y_k^{i_M K}, W_k^{i_M K})$, and J_{k+1} can be placed (early) in superinterval $G_{i_{u-1}i_u}$ by our algorithm.

Case 2: J_{k+1} is tardy. Since J_{k+1} is tardy, $t \le i_{u-1}$. Also, $L_k^{0l*} - L_k^{0i_{u-1}*} \ge p_{k+1}$ holds, $\forall l$ s.t. $t \le l \le i_u$. Therefore we have $L_k^{0l*} \ge L_k^{0l*} - L_k^{0i_{u-1}*} \ge p_{k+1}$. If $L_k^{0l} - L_k^{0i_{u-1}} \ge p_{k+1}$ holds $\forall l$ s.t. $t \le l \le i_u$, then Condition (1a) is satisfied. Otherwise, we examine Condition (1b): We have $L_k^{0l} \ge L_k^{0l*} \ge L_k^{0l*} - L_k^{0i_{u-1}*} \ge p_{k+1}\ \forall l$, s.t. $i_{u-1} < l \le i_u$. Additionally, Condition (1c) is satisfied, because $L_k^{0i_j} \ge L_k^{0i_j*} \ge p_{k+1},\ \forall j$, s.t. $u < j \le M$. Hence, Condition (1) is satisfied by state $(k, Z_k, y_k^{i_1 1}, W_k^{i_1 1}, \cdots, y_k^{i_M K}, W_k^{i_M K})$, and J_{k+1} can be placed (tardy) in superinterval $G_{i_{u-1}i_u}$ by our algorithm.

□

In the rest of the paper, we work with these two special sequences and their transitions. We observe that $L_m^{0j*} \ge \sum_{h=1}^{u-1} p_{S_{i_h}}\ \forall j, u$ s.t. $1 < u \le M$ and $i_{u-1} < j \le i_u$ from Condition (3), which is satisfied by the optimal DP. Moreover, $L_m^{0l} \ge L_m^{0l*}\ \forall l$ s.t. $1 \le l \le K$ (cf. Lemma 10). Hence $L_m^{0j} \ge \sum_{h=1}^{u-1} p_{S_{i_h}}\ \forall j, u$ s.t. $1 < u \le M$ and $i_{u-1} < j \le i_u$, 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 every $1 \le k \le m$ and $1 \le u \le M$, let $B_{i_u}^*(k) := \max\{w_h \mid k \le h \le m, y_h^{i_u j} \ne 0, 1 \le j \le i_u\}$, and if no job is tardy in superinterval $G_{i_u i_{u+1}}$, set $B_{i_u}^*(k) := 0$.

Lemma 11 *For every $1 \le k \le m$, $1 \le u \le M$, and $1 \le j \le i_u \le K$ we have*

$$Z_k \le Z_k^* + 2k\delta \tag{6}$$

$$0 \le i_u B_{i_u}^*(k)(y_k^{i_u j} - y_k^{i_u j*}) \le \delta \tag{7}$$

Proof: The proof by induction is essentially the same as the proof of Lemma 1 in [4], but we include it here for completeness. Assume that J_{k+1} is from C_t where $1 \le t \le K$, and to be inserted in superinterval $G_{i_u i_{u+1}}$. Then $Z_{k+1}^* = Z_k^*$ if J_{k+1} is early, and $Z_{k+1}^* = Z_k^* + w_{k+1}(\sum_{j=1}^{i_u} y_k^{i_u j^*} + p_{k+1} + d_{i_u} - d_t)$ if it is tardy. 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 $k = 1$, recall that the initial state is $(0, 0, \ldots, 0)$. It is easy to verify (6),(7) by the definition of the rounding. Now assume that these conditions hold for $k = s$, where $s < m$. We prove the lemma for $k = s + 1$.

If, in the optimal sequence, $y_{s+1}^{i_u t*} = y_s^{i_u t*} + p_{s+1} > 0$, then J_{s+1} is tardy in $G_{i_u i_{u+1}}$ and hence $B_{i_u}^*(s+1) > 0$. Assume that $B_{i_u}^*(s+1) = w_v$ 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 $y_{s+1}^{i_u t*} \leq \frac{Z^{ub}}{B_{i_u}^*(s+1)}$, and $y_{s+1}^{i_u t*}$ belongs to some subinterval of length at most $\frac{\delta}{i_u B_{i_u}^*(s+1)}$.

Now Lemma 10 implies that J_{s+1} is placed early or tardy by both the optimal and our sequence, but in both cases 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 (7).

Now we are back to prove (6). If J_{s+1} is inserted early, the result is trivial. If it is tardy, we have

$$
\begin{aligned}
Z_{s+1} &\leq Z_s + w_{s+1}\left(\sum_{j=1}^{i_u} y_s^{i_u j} + p_{s+1} + d_{i_u} - d_t\right) + \delta \\
&\overset{(6),(7)}{\leq} Z_s^* + 2s\delta + w_{s+1}\left(\sum_{j=1}^{i_u}(y_s^{i_u j*} + \frac{\delta}{i_u B_{i_u}^*(s+1)}) + p_{s+1} + d_{i_u} - d_t\right) + \delta \\
&= Z_s^* + w_{s+1}\left(\sum_{j=1}^{i_u} y_s^{i_u j*} + p_{s+1} + d_{i_u} - d_t\right) + i_u w_{s+1}\frac{\delta}{i_u B_{i_u}^*(s+1)} + \delta + 2s\delta \\
&\leq Z_{s+1}^* + 2(s+1)\delta
\end{aligned}
$$

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 [4], and we include it here for completeness purposes.

Theorem 2 *If Z is the total tardiness of the schedule returned by the algorithm and Z^* is the optimal, we have that $Z \leq (1+\varepsilon)Z^*$.*

Proof: In exactly the same way as in the proof of Lemma 2 of [4], we can show that

$$
W_m^{i_u j*} \leq W_m^{i_u j} \leq (1+\frac{\varepsilon}{2}) W_m^{i_u j*},\ \forall u, j \text{ s.t. } 1 \leq u \leq M,\ 1 \leq j \leq i_u. \tag{8}
$$

Let Z_m be the total tardiness of the partial schedule computed by the algorithm before inserting the straddlers. Recall from (4) that $x_{i_u} := \max\{0, \sum_{l=1}^{u} p_{S_{i_l}} - L_m^{0 i_u}\}$ (and respectively for $x_{i_u}^*$).

Since $L_m^{0i_{u^*}}$ is rounded up, $x^*_{i_u}$ is rounded down (or becomes 0), i.e., $x_{i_u} \leq x_{i_{u^*}}$. Then we have

$$
\begin{aligned}
Z &\overset{(5)}{=} Z_m + \sum_{u=1}^{M} w_{i_u} T_{i_u} + \sum_{u=1}^{M} (\sum_{l=1}^{i_u} W_m^{i_u l}) x_{i_u} \\
&= Z_m + \sum_{u=1}^{M} w_{i_u} (x_{i_u} + d_{i_u} - d_t) + \sum_{u=1}^{M} (\sum_{l=1}^{i_u} W_m^{i_u l}) x_{i_u} \\
&\leq Z_m + \sum_{u=1}^{M} w_{i_u} (x^*_{i_u} + d_{i_u} - d_t) + \sum_{u=1}^{M} (\sum_{l=1}^{i_u} W_m^{i_u l}) x^*_{i_u} \\
&\overset{(6),(8)}{\leq} Z^*_m + 2m\delta + \sum_{u=1}^{M} w_{i_u} (x^*_{i_u} + d_{i_u} - d_t) + (1 + \frac{\varepsilon}{2}) \sum_{u=1}^{M} (\sum_{l=1}^{i_u} W_m^{i_u l*}) x^*_{i_u} \\
&\leq Z^*_m + \frac{\varepsilon}{2} Z_{lb} + \sum_{u=1}^{M} w_{i_u} (x^*_{i_u} + d_{i_u} - d_t) + \sum_{u=1}^{M} (\sum_{l=1}^{i_u} W_m^{i_u l*}) x^*_{i_u} + \frac{\varepsilon}{2} \sum_{u=1}^{M} (\sum_{l=1}^{i_u} W_m^{i_u l*}) x^*_{i_u} \\
&\overset{(5)}{\leq} Z^* + \frac{\varepsilon}{2} Z^* + \frac{\varepsilon}{2} Z^* = (1 + \varepsilon) Z^*
\end{aligned}
$$

□

General straddler placement: Recall that till now we have assumed that each one of the (guessed) tardy straddlers straddles only one due date. From the above, it is easy to see how the algorithms can be modified to work for the general case of straddlers spanning over more than one due date.

Acknowledgment

We thank George Steiner for several enlightening discussions.

References

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

[2] T. C. E. Cheng, C. T. Ng, J. J.Yuan, Z. H. Liu. Single machine scheduling to minimize total weighted tardiness. *European J. Oper. Res.,* Vol. 165, pp. 423–443, 2005

[3] J. Du and J. Y.-T. Leung. Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research,* Vol. 15, pp. 483–495, 1990.

[4] 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,* Vol. 369, pp. 230–238, 2006.

[5] S. G. Kolliopoulos and G. Steiner. Approximation Algorithms for Minimizing the Total Weighted Tardiness on a Single Machine. *Theoretical Computer Science,* Vol. 355(3), pp. 261–273, 2006.

[6] E. L. Lawler. A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. *Ann. Discrete Math.,* Vol. 1, pp. 331–342, 1977.

[7] E .L. Lawler. A fully polynomial approximation scheme for the total tardiness problem. *Operations Research Letters,* Vol. 1, pp. 207–208, 1982.

[8] J. K. Lenstra, A. H. G. Rinnooy Kan, P.Brucker. Complexity of machine scheduling problems. *Ann. Discrete Math.,* Vol. 1, pp. 343–362, 1977.

[9] R. McNaughton. Scheduling with due dates and loss functions, *Management Sci.* Vol. 6, pp. 1–12, 1959.

[10] T. Sen, J. M. Sulek, and P. Dileepan. Static scheduling research to minimize weighted and unweighted tardiness: a state-of-the-art survey. *Int. J. Production Econom.,* Vol. 83, pp. 1–12, 2003.

[11] J. Yuan. The NP-hardness of the single machine common due date weighted tardiness problem. *Systems Sci. Math. Sci.,* Vol. 5, pp. 328–333, 1992.