Approximation algorithms for minimizing the total weighted tardiness on a single machine

Stavros G. Kolliopoulos* George Steiner†

Abstract

Given a single machine and a set of jobs with due dates, the classical $\mathcal{NP}$-hard problem of scheduling to minimize total tardiness is a well-understood one. Lawler gave an FPTAS for it some twenty years ago. If the jobs have positive weights the problem of minimizing total weighted tardiness seems to be considerably more intricate. In this paper, we give some of the first approximation algorithms for it. We examine first the weighted problem with a fixed number of due dates and we design a pseudopolynomial algorithm for it. We show how to transform the pseudopolynomial algorithm to an FPTAS for the case where the weights are polynomially bounded.

For the case with an arbitrary number of due dates and polynomially bounded processing times, we provide a quasipolynomial algorithm which produces a schedule whose value has an additive error proportional to the weighted sum of the due dates. We also investigate the performance of algorithms for minimizing the related total weighted late work objective.

1 Introduction

We study the problem of scheduling jobs on a single machine to minimize the total weighted tardiness. We are given a set of n jobs. Job j, $1 \leq j \leq n$, becomes available at time 0, has to be processed without interruption for an integer time p_j, has a due date d_j, and has a positive weight w_j. For a given sequencing of the jobs the *tardiness* T_j of job j is defined as $\max\{0, C_j - d_j\}$, where C_j is the completion time of the job. The objective is to find a processing order of the jobs which minimizes $\sum_{j=1}^{n} w_j T_j$. In the 3-field notation used in scheduling the problem is denoted $1||\sum_j w_j T_j$.

According to Congram et al., $1||\sum_j w_j T_j$ is an "$\mathcal{NP}$-hard archetypal machine scheduling problem" whose exact solution appears very difficult even on very small inputs [2]. We proceed to review what is known on the complexity of the problem. In the case of one machine it has long been known that an optimal preemptive schedule has the same total weighted tardiness as an optimal nonpreemptive schedule [13]. Early on the problem was shown to be $\mathcal{NP}$-hard in the ordinary

*Department of Informatics and Telecommunications, National and Kapodistrian University of Athens; (www.di.uoa.gr/~sgk). Research supported in part by the program EPEAK II under the task PYTHAGORAS II (project title: *Algorithms and Complexity in Network Theory*) which is funded by the European Social Fund (75%) and the Greek Ministry of Education (25%).

†Management Science and Information Systems, McMaster University, 1280 Main Street West, Hamilton, Ontario, L8S 4M4, Canada. Research partially supported by NSERC Grant OG0001798; Corresponding author (steiner@mcmaster.ca).

sense by Lenstra et al. [12] when the jobs have only two distinct due dates by a reduction from the knapsack problem. It was shown to be strongly $\mathcal{NP}$-hard for an arbitrary number of due dates in [9]. Much later Yuan [15] showed that the problem remains $\mathcal{NP}$-hard even for the case where all the jobs have a common due date. Lawler and Moore [11] have presented a pseudopolynomial solution for the case when all jobs have a single common due date. From the approximation point of view there is very little known. The only case that seems to be better understood is the usually easier case of *agreeable weights:* in that case $p_j < p_i$ implies $w_j \geq w_i$. Lawler gave a pseudopolynomial algorithm for the agreeable-weighted case [9]. In 1982 he showed how to modify that algorithm to obtain a fully polynomial-time approximation scheme for the case of unit weights [10]. A fully polynomial-time approximation scheme (FPTAS) for a minimization problem is an algorithm which for any $\varepsilon > 0$ runs in time polynomial in $1/\varepsilon$ and the size of the input and outputs a solution of cost at most $(1+\varepsilon)$ times the optimum. After the first publication of this work [7], the paper by Cheng, Ng, Yuan and Liu appeared [1]. There it is shown that the schedule which minimizes $\max_j w_j T_j$ yields an $(n-1)$-approximation for $1|\,|\sum_j w_j T_j$. Interestingly, the complexity of the unit weight case, $1|\,|\sum_j T_j$, was an open problem for many years until Du and Leung showed it is $\mathcal{NP}$-hard [3].

In this paper we make progress on the problem of minimizing the total weighted tardiness by examining first the case where the number of distinct due dates is fixed. Our main contribution is a pseudopolynomial algorithm whose complexity depends on the total processing time. This implies that the problem is in $\mathcal{P}$ when the processing times are polynomially bounded. We then show how to modify the pseudopolynomial algorithm in two steps: first so that its complexity depends on any upper bound on the maximum tardiness of an optimal schedule and second, so that it yields an FPTAS when the maximum job weight is bounded by a polynomial in n. Our main approach is based on viewing the problem as having to pack the jobs into a finite number of bins where the cost of each job depends on which bin it is assigned to and some jobs may be split between two bins. Hopefully some of the ideas we introduce could be of use for further study of approximating the long-open general case with an arbitrary number of due dates.

For the general case with an arbitrary number of distinct due dates we give a result that may be of interest when the due dates are concentrated around small values. Under the assumption that the maximum processing time is bounded by a polynomial in n, we provide a quasipolynomial algorithm which produces a schedule whose value has an additive error equal to $\delta \sum_j w_j d_j$ for any fixed $\delta > 0$. $1|\,|\sum_j w_j T_j$ falls into the class of $\mathcal{NP}$-hard problems where the optimum value can be zero. This renders the notion of a multiplicative approximation somewhat problematic. Additive guarantees may be of particular interest in this setting. We obtain the latter result by combining a partition of the time horizon into geometrically increasing intervals with a shift of the due dates to the interval endpoints.

The total weighted late work is an objective function which is conceptually related to the total weighted tardiness. The *late work* V_j of job j in a given schedule is defined as $\min\{T_j, p_j\}$. In other words, the late work is the amount of processing performed on job j after its due date d_j. Hence $V_j \leq T_j$, with equality in the case where the start time of job j is on or before d_j. The problem of minimizing the *total weighted late work*, denoted by $1|\,|\Sigma_j w_j V_j$, is a well-studied problem which is $\mathcal{NP}$-hard even with unit weights [14]. We formalize the relation between the two objective functions by showing that an optimal schedule for $1|\,|\sum_j w_j V_j$ achieves an $O(P)$ approximation with respect to the total weighted tardiness, where $P \doteq \sum_{i=1}^{n} p_i$ is the total processing time. We find it interesting that the guarantee is independent of the weights. We survey the known

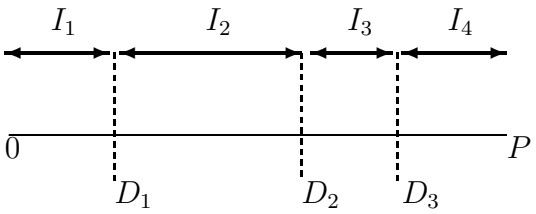

Figure 1: Intervals

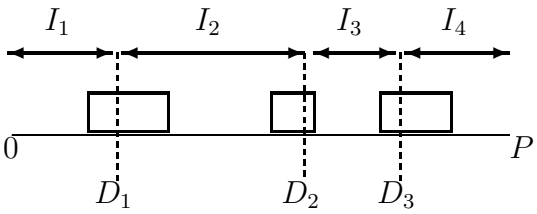

Figure 2: Placement of straddlers

algorithmic results on $1|\,|\sum_j w_j V_j$ in Section 4.2.

The outline of this paper is as follows. In Section 2 we present two pseudopolynomial algorithms. A first relatively simple one is presented in Section 2.2. The more involved one in Section 2.4 is shown to be amenable to a transformation to an FPTAS in Section 3. The quasipolynomial algorithm is presented in Section 4.1. In Section 4.2 we examine schedules that minimize the total late work from the point of view of total weighted tardiness. We conclude with open questions in Section 5. This paper is an extensively revised and augmented version of [7].

2 Pseudopolynomial algorithms

2.1 Preliminary remarks

The m distinct due dates $D_1 < D_2 < \ldots < D_m$, partition the time horizon $[0, P]$ into $m+1$ intervals $I_1, I_2, \ldots, I_{m+1}$ where $I_1 = [0, D_1]$, $I_i = (D_{i-1}, D_i]$, $2 \leq i \leq m$ and $I_{m+1} = (D_m, P]$. See Fig. 1 for an example. A job is *early* in a schedule if its processing is completed by its due date, and a job is *tardy* if it completes after its due date. We now define a special type of jobs that plays a central role in our algorithms.

Definition 2.1 *We call a job j the i*th straddler, *$i = 1, \ldots, m$ if (1) j is the last job to start in the interval I_i and (2) j finishes after D_i.*

It could be that no ith straddler exists. This could happen if the last job to start in I_i finishes exactly at D_i or if the jth straddler for some $1 \leq j < i$ has not completed by D_i. If no job meets the conditions for being the ith straddler we say that the latter's value is Φ. As it is observed in [11], there appears to be no way to identify the straddling jobs in an optimal schedule before finding that schedule. After the identity of the straddlers and their respective starting times have been guessed, we need to *pack* the rest of the jobs in the remaining space within the intervals. See Fig. 2 for an example. This packing approach where the intervals play the role of bins guides the design of our algorithms.

If one decides that some set A of jobs will be scheduled consecutively as a continuous block of tardy jobs, they need to minimize $\sum_{j \in A} w_j C_j - \sum_{j \in A} w_j d_j$. The second term is independent of the ordering of the jobs, therefore it suffices that the order minimizes the sum of the weighted completion times. This suggests that we can assume that the jobs have been numbered in *weighted shortest processing time* (WSPT) order, i.e., $p_1/w_1 \leq p_2/w_2 \leq ... \leq p_n/w_n$.

Lemma 2.1 *In any optimal schedule the non-straddling tardy jobs scheduled consecutively in any interval $I_i (i > 1)$ must appear in WSPT order.*

Proof. Let J_i be the set of non-straddling tardy jobs scheduled consecutively in I_i in an optimal schedule. Assume that the jobs in J_i do not follow the WSPT order. Then there exist two adjacent jobs $j, l \in J_i$ such that $p_j/w_j < p_l/w_l$, but j is scheduled in the position immediately following l. A simple interchange argument shows that switching j in front of l would reduce the total weighted tardiness of the schedule, which contradicts its optimality. ∎

Another observation which will be of use is due to McNaughton [13], we include its easy proof here for the sake of completeness.

Lemma 2.2 *The preemptive and non-preemptive versions of any instance of the total weighted tardiness problem have the same minimum total weighted tardiness.*

Proof. Consider any optimal preemptive schedule. Take all but the last piece of a job and insert these immediately before its last piece while shifting every other job as much to the left as possible. The total weighted tardiness of the resulting schedule is not worse than that of the original schedule. Repeat this operation for every preempted job. ∎

The intervals in which a job j can be early or tardy are determined by the value of d_j. This motivates the following definition. We say that job j *belongs to job class* $\mathcal{C}_r$ if $d_j = D_r$ for $r \in \{1, 2, \ldots, m\}$. A job from $\mathcal{C}_r$ with completion time in the intervals $I_1, I_2, \ldots, I_r$ is early; with completion time in the intervals $I_{r+1}, I_{r+2}, \ldots, I_{m+1}$ it is tardy. Within a single interval it is obvious that all the tardy jobs must precede the early ones irrespective of class. By Lemma 2.1 the ordering within the block of tardy jobs should conform to WSPT.

2.2 A first pseudopolynomial algorithm

In this section we present the first of two different dynamic programming algorithms. This algorithm is simpler compared to the one in Section 2.4 in that it considers only nonpreemptive schedules. Therefore it corresponds directly to the intuition of packing jobs in intervals. It has the disadvantage that it is not amenable to the transformation to an FPTAS described in Section 3.

For m distinct due dates the time horizon is partitioned into $m+1$ intervals $I_1, I_2, \ldots, I_{m+1}$ as described. Let $S = \{k_1, k_2, \ldots, k_m\}$ denote the set of the m straddlers where some k_i may take the value Φ. In that case the starting time S_{k_i} also takes the value Φ. Variable t_i, $i = 1, \ldots, m+1$ stands for the total processing time of tardy jobs, excluding the straddlers, in the interval I_i. Variable e_i, $i = 1, \ldots, m+1$ stands for the total processing time of early jobs, excluding the straddlers, in I_i. By definition $t_1 = e_{m+1} = 0$. Since jobs scheduled within the e_i part are early anywhere in I_i, it is clear that jobs within the t_i part should precede the jobs in e_i. Recall that the jobs are numbered in order of non-decreasing p_i/w_i ratio.

Given the set S, the tuple describing a state of the dynamic program is

$$(S_{k_1}, S_{k_2}, \ldots, S_{k_m}, e_1, \ldots, e_m, t_2, \ldots, t_{m+1}, j)$$

where $j \notin S$ corresponds intuitively to the last job in the WSPT ordering which has been scheduled. The special value $j = 0$, means that no job has been scheduled yet. For a given tuple x in the state space $C(x)$ denotes the cost, i.e., the corresponding minimum weighted tardiness value. A tuple is called *legal* if the values $S_{k_1}, S_{k_2}, \ldots, S_{k_m}$ $e_1, \ldots, e_m, t_2, \ldots, t_{m+1}$ and j are compatible from a packing perspective with each other. This means that the following three conditions are satisfied.

Legality Condition 1. Long straddlers occupy space:
$k_i = \Phi$ and no job starts in $I_i \Rightarrow e_i = t_i = 0, \;\; i > 1$.

Legality Condition 2. The length of each interval is not exceeded:
$e_i + t_i + D_i - S_{k_i} \leq D_i - D_{i-1} - A_i \;\; \forall k_i \neq \Phi$ and $i \geq 2$

The quantity A_i denotes the space taken in I_i by the previous straddler before k_i and is defined as follows.

Case 1: there is a k_j s.t. $1 \leq j \leq i-1$ and $k_j \neq \Phi$ and $C_{k_j} \geq D_{i-1}$; i.e., there is a straddler that finishes at or after D_{i-1} (and before D_i since $k_i \neq \Phi$). Then $A_i \doteq C_{k_j} - D_{i-1}$, i.e., A_i equals the space taken in I_i by this straddler. *Case 2:* otherwise, $A_i \doteq 0$.

Legality Condition 3. There is enough processing time for all the jobs seen so far:
$\sum_{i=1,\ldots,j | i \notin S} p_i = \sum_{k=1,\ldots,m} e_k + \sum_{l=2,\ldots,m+1} t_l$.

In the remainder of this section we consider only legal tuples.

The dynamic program enumerates the possible choices for the m straddlers and, once this is fixed their possible starting times. For every prefix of the sequence of jobs $1, 2, \ldots, n$ with the straddlers removed it enumerates the possible values for the variables e_i, t_i that allow a placement of the current job, i.e., the job at the end of the prefix. Let us assume then that a choice has been made for the set of straddlers and their placement. Let $INIT \doteq \sum_{k_i \neq \Phi} w_{k_i} \max\{0, S_{k_i} + p_{k_i} - d_{k_i}\}$. The initialization is done as follows.

Initial Condition: For all legal tuples $\mathcal{T}$ of the form $(S_{k_1}, S_{k_2}, \ldots, S_{k_m}, 0, \ldots, 0, 0, 0)$, set $C(\mathcal{T}) \doteq INIT$.

For ease of exposition we give a recursive implementation of the algorithm. Let the current job in the WSPT ordering be $j \notin S$. The last job before it in the ordering which is not a straddler is denoted as $last(j) \doteq \max(\{0, 1, 2, ..., j-1\} \setminus S)$. Assume that job j belongs to class $\mathcal{C}_r$, $r = 1, \ldots, m$. Recall that this means that $d_j = D_r$. Job j can be early in the intervals $I_1, I_2, \ldots, I_r$ and tardy in the intervals $I_{r+1}, I_{r+2}, \ldots, I_{m+1}$. Let $\mathcal{T} = (S_{k_1}, S_{k_2}, \ldots, S_{k_m}, e_1, \ldots, e_m, t_2, \ldots, t_{m+1}, j)$. $C(\mathcal{T})$ is set equal to the minimum of the following $r + m + 1 - r = m + 1$ quantities. It is easy to define an order of computation so that the values needed are available. The value T_j is defined later.

$C\,(S_{k_1}, S_{k_2}, \ldots, S_{k_m}, e_1 - p_j, \ldots, e_m, t_2, \ldots, t_{m+1}, last(j))$ if $e_1 \geq p_j$
$C(S_{k_1}, S_{k_2}, \ldots, S_{k_m}, e_1, e_2 - p_j, \ldots, e_m, t_2, \ldots, t_{m+1}, last(j))$ if $e_2 \geq p_j$
$\ldots$
$C(S_{k_1}, S_{k_2}, \ldots, S_{k_m}, e_1, \ldots, e_r - p_j, \ldots, e_m, t_2, \ldots, t_{m+1}, last(j))$ if $e_r \geq p_j$
$C(S_{k_1}, S_{k_2}, \ldots, S_{k_m}, e_1, \ldots, e_m, t_2, t_3, \ldots, t_{r+1} - p_j, t_{r+2}, \ldots, t_{m+1}, last(j)) + w_j T_j$ if $t_{r+1} \geq p_j$
$C(S_{k_1}, S_{k_2}, \ldots, S_{k_m}, e_1, \ldots, e_m, t_2, t_3, \ldots, t_{r+1}, t_{r+2} - p_j, \ldots, t_{m+1}, last(j)) + w_j T_j$ if $t_{r+2} \geq p_j$
$\ldots$
$C(S_{k_1}, S_{k_2}, \ldots, S_{k_m}, e_1, \ldots, e_m, t_2, t_3, \ldots, t_{r+1}, t_{r+2}, \ldots, t_{m+1} - p_j, last(j)) + w_j T_j$ if $t_{m+1} \geq p_j$

Each case is accompanied by an if condition that declares what is necessary for j to be able to "fit" in the corresponding interval. If one such condition is not met the corresponding branch is considered to yield the value ∞. If none of the $m + 1$ if conditions is satisfied $C(\mathcal{T})$ is set to ∞.

The first r cases correspond to all possible placements of job j as early. The last $m + 1 - r$ cases correspond to j being tardy. If j is considered for being tardy in interval I_l, $l \geq 2$, extra care must be taken since the straddler k_{l-1} may be undefined. Therefore when computing T_j in I_l it cannot always be expected to equal $S_{k_{l-1}} + p_{k_{l-1}} + t_l - D_r$. T_j is computed as follows:

Case 1: there is a k_i s.t. $k_i \neq \Phi$ and $C_{k_i} \geq D_{l-1}$ i.e., there is a straddler finishing at or after

D_{l-1}. Then $T_j := S_{k_i} + p_{k_i} + t_l - D_r$. *Case 2:* $k_{l-1} = \Phi$ and Case 1 does not apply. Then $T_j := D_{l-1} + t_l - D_r$.

The time complexity of the dynamic programming algorithm will be $O(m\Lambda)$ where Λ is the size of the state space. Given that at most all of the m straddlers may be Φ, the number of ways to fill up the m straddler slots is at most $m!\binom{n+m}{m}$. Given the set S, let $1 \leq l \leq m$ be the number of straddlers that are different from Φ. The number of possible placements of the straddlers with respect to the due dates $D_1, D_2, \ldots, D_m$ is at most $\Pi_{k_i \neq \Phi} p_{k_i} \leq (\sum_{k_i \neq \Phi} p_{k_i}/l)^l \leq (P/l)^l$. For the second inequality we used the well-known relation between geometric and arithmetic mean, see for instance [4]. The quantity $(P/l)^l$ is maximized when $l = m$. Therefore the maximum contribution of the straddlers to the size of the state space is a multiplicative $m!\binom{n+m}{m}(P/m)^m$ factor. The sum of all the e_i, t_i variables cannot exceed P. Therefore the number of interesting tuples of the form $(e_1, \ldots, e_m, t_2, \ldots, t_{m+1})$ is upperbounded by the quantity

$$\prod_{(x_1, x_2, \ldots, x_{2m}) \in \Gamma} x_i, \qquad \text{where } \Gamma = \{(x_1, x_2, \ldots, x_{2m}) \in \mathbb{Z}_+^{2m} : x_1 + \ldots + x_{2m} \leq P\}.$$

Applying as above the fact that the geometric mean is at most the arithmetic mean, we obtain that the number of interesting tuples of the form $(e_1, \ldots, e_m, t_2, \ldots, t_{m+1})$ is $O((P/m)^{2m})$. Therefore $\Lambda = O(m!\binom{n+m}{m}(P/m)^{3m}n)$. The following theorem has been shown.

Theorem 2.1 *There is an algorithm with complexity* $O\left(m(m!\binom{n+m}{m}(P/m)^{3m}n)\right)$ *which computes the minimum total weighted tardiness for a problem with* m *distinct due dates.*

2.3 The state space of the second dynamic program

The second dynamic programming algorithm allows *preemption of the early jobs.* Lemma 2.2 showed that this is safe. We proceed to define the state space of the algorithm. Again let $S = \{k_1, k_2, \ldots, k_m\}$ denote the set of the m straddlers where some k_i may take the value Φ. Let $S_{k_1}, S_{k_2}, \ldots, S_{k_m}$ be the set of the corresponding start times. If $k_i = \Phi$ the starting time S_{k_i} also takes the value Φ. From now on we call *straddlers the jobs that meet Definition 2.1 and in addition are tardy.* The reason for this new definition is that we want Theorem 2.3 to hold, which will be useful when designing an FPTAS in Section 3. We want the size of the state space of the new dynamic program to be independent of P. Allowing non-tardy straddlers seems to make this impossible.

In particular, a key requirement for obtaining an FPTAS out of a dynamic program in Section 3 is that we can express the values of our state space variables as functions of a suitable upper bound on the maximum tardiness in a schedule σ^* which optimizes $\sum_j w_i T_j$. This maximum tardiness is not only instance-dependent but it also depends on the specific optimal schedule σ^*. The upper bound we will use in Section 3, however, holds for any σ^*. Therefore we construct the dynamic program in Section 2.4 by considering a specific optimal schedule σ^* without worrying about how to choose it.

An early job that starts before D_i and completes after D_i will be considered by our algorithm as being preempted at time D_i and being immediately restarted.

For each interval $I_2, I_3, \ldots, I_{m+1}$ we define an m-tuple τ_i that breaks down the information on tardy jobs across job classes.

$$\tau_i = (t_{i1}, t_{i2}, \ldots, t_{im}),\ i = 2, \ldots, m$$

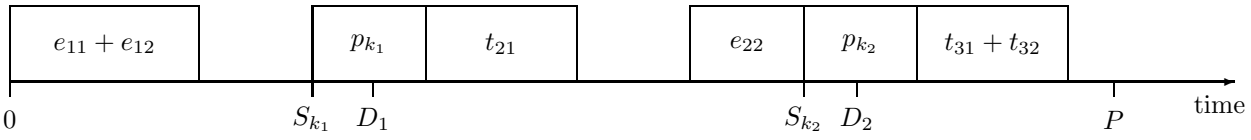

Figure 3: The Gantt chart for a partial schedule with two due dates. The quantity e_{ij} denotes the total processing time of early jobs from class j which are scheduled in interval I_i.

where t_{ij} is the total length of tardy jobs from class j scheduled in interval I_i. The tuple τ_i induces the work variable $t_i = \sum_{r=1}^{m} t_{ir}$. As in Section 2.2 t_i stands for the total processing time of tardy jobs, excluding the straddlers, in interval I_i. The difference is that we do not need to enumerate the values of the t_is, the τ_i tuples suffice. Observe that always $t_{ir} = 0$ for $r \geq i$. See Fig. 3 for an example Gantt chart of a partial schedule.

Recall that the jobs are numbered in order of non-decreasing p_i/w_i ratio. Given the set S, the tuple describing a state of the dynamic program is

$$(S_{k_1}, S_{k_2}, \ldots, S_{k_m}, \tau_2, \ldots, \tau_{m+1}, j)$$

where $j \notin S$ is considered to be the last job in the WSPT ordering which has been processed, i.e., scheduled. A tuple is called *legal* if the values $S_{k_1}, S_{k_2}, \ldots, S_{k_m}$ are compatible with the induced values $t_2, \ldots, t_{m+1}$ and with j. The following three conditions need to be satisfied.

Legality Condition 1. Long straddlers occupy space:
(If $k_i = \Phi$ and no job starts in I_i) $\Rightarrow t_i = 0, \;\; i > 1$.

Legality Condition 2. The length of each interval is not exceeded:
$t_i + D_i - S_{k_i} \leq D_i - D_{i-1} - A_i \;\; \forall k_i \neq \Phi$ and $i \geq 2$

The quantity A_i equals the space taken in I_i by the previous straddler before k_i and is defined as in Section 2.2.

To express the third legality condition we need first to extract from a tuple

$$\mathcal{T} = (S_{k_1}, S_{k_2}, \ldots, S_{k_m}, \tau_2, \ldots, \tau_{m+1}, j)$$

information on the length of the early jobs. Let $P_r^j \doteq \sum_{i \in \mathcal{C}_r \setminus S, i \leq j} p_i$, $r = 1, \ldots, m$. Define $E_r(\mathcal{T}) = P_r^j - \sum_{i=2}^{m+1} t_{ir}$. This quantity denotes the total length of early jobs from $\mathcal{C}_r \cap \{1, 2, \ldots, j\} \setminus S$, $r = 1, \ldots, m$.

Legality Condition 3. There is enough processing time for all the jobs seen so far:
$\sum_{i=1,\ldots,j | i \notin S} p_i = \sum_{r=1,\ldots,m} E_r(\mathcal{T}) + \sum_{l=2,\ldots,m+1} t_l$.

In the remainder of this section we consider only legal tuples.

The dynamic program enumerates the possible choices for the m straddlers and, once this is fixed, their possible starting times. For every prefix of the sequence of jobs $1, 2, \ldots, n$ with the straddlers removed it enumerates the possible values for the variables t_i, τ_i that allow a placement of the current job, i.e., the job at the end of the prefix. Let us assume then that a choice has been made for the set of straddlers and their placement. Let $INIT \doteq \sum_{k_i \neq \Phi} w_{k_i}(S_{k_i} + p_{k_i} - d_{k_i})$. The initialization is done as follows and it corresponds to states expressing a placement of the straddlers without any other job having been processed.

Initial Condition: For all legal tuples $\mathcal{T}$ of the form $(S_{k_1}, S_{k_2}, \ldots, S_{k_m}, 0, \ldots, 0, 0)$, set $C(\mathcal{T}) \doteq INIT$.

2.4 The second pseudopolynomial algorithm

For ease of exposition we give a recursive implementation of the algorithm. Each state as defined above encodes some information about the scheduling of the early jobs as well. Recall the definition of the E_r values in the previous section. The algorithm needs to decode this information on the fly to avoid increasing the size of the state space. The reason we compress the information has to do with the technique we use later in Section 3 to transform the dynamic program into an FPTAS. In contrast, explicit information on the total length of the early jobs in each interval was maintained by the dynamic program in Section 2.2.

Fix a tuple

$$\mathcal{T} = (S_{k_1}, S_{k_2}, \ldots, S_{k_m}, \tau_2, \ldots, \tau_{m+1}, j)$$

of the state space. Observe that the early jobs of a class can be spread in more than one interval. By Lemma 2.2 a single job may be split across different intervals and this does not affect the optimum. This observation allows us to reason about the early jobs using the limited information provided by the E_r values.

Consider the tuple $\mathcal{T}$ defined above and assume that job j belongs to class $\mathcal{C}_r$, for some $r \in \{1, \ldots, m\}$. Recall that job j can be early in the intervals $I_1, I_2, \ldots, I_r$ and tardy in the intervals $I_{r+1}, I_{r+2}, \ldots, I_{m+1}$. We want to determine the cost $C(\mathcal{T})$, i.e., the total minimum weighted tardiness corresponding to this state. We compute

$$C(\mathcal{T}) = \min\{A, B\},$$

where A corresponds to the value of the best placement of j as early and B to the best placement of j as tardy. We now proceed to define A and B. Recall the definition of $last(j)$ from Section 2.2.

Early case: A is equal to the minimum value $C(\mathcal{T}')$ over all tuples $\mathcal{T}'$ of the form $\big(S_{k_1}, S_{k_2}, \ldots, S_{k_m}, \tau'_2, \ldots, \tau'_{m+1}, last(j)\big)$ that meet the following two conditions:

1. For all $i = 2, \ldots, m+1$, the tuples τ_i, τ'_i, agree on all coordinates.
2. $E_r(\mathcal{T}') = E_r(\mathcal{T}) - p_j \geq 0$.

Condition 1 implies that during the state transition no change occurs on the total length of tardy jobs. Condition 2 implies that there is enough available total space in the intervals where job j can be scheduled as early.

Tardy case: B is equal to the minimum value $C(\mathcal{T}') + w_j T_j$ over all tuples $\mathcal{T}'$ of the form $(S_{k_1}, S_{k_2}, \ldots, S_{k_m}, \tau_2, \ldots, \tau_{l-1}, \tau'_l, \tau_{l+1}, \ldots, \tau_{m+1}, last(j))$ that meet the following two conditions:

1. $l \in \{r+1, r+2, \ldots, m+1\}$.
2. $\tau'_l = (t_{l1}, \ldots, t_{l(r-1)}, t_{lr} - p_j, t_{l(r+1)}, \ldots, t_{lm})$ and $t_{lr} - p_j \geq 0$.

In words, $\mathcal{T}'$ is a tuple with enough empty space in the tardy portion of one of the intervals $I_{r+1}, \ldots, I_{m+1}$ for j to be scheduled. Moreover, during the state transition no change occurs on

the total length of jobs belonging to classes different from $\mathcal{C}_r$.
Once j is assigned at the end of the tardy portion of interval I_l, its tardiness depends on the total length t_l of tardy jobs and the completion time of the straddler preceding them. T_j is computed as follows:
Case 1: there is a k_i s.t. $k_i \neq \Phi$ and $C_{k_i} \geq D_{l-1}$, i.e., there is a straddler finishing at or after D_{l-1}. Then $T_j := S_{k_i} + p_{k_i} + t_l - D_r$. *Case 2:* $k_{l-1} = \Phi$ and Case 1 does not apply, i.e., the first tardy job in I_l starts at D_{l-1}. Then $T_j := D_{l-1} + t_l - D_r$.

If no state can meet the conditions defined in either of the two cases, we set $c(\mathcal{T})$ to ∞. Given a set S of straddlers, the minimum total weighted tardiness of complete schedules with these straddlers is equal to the minimum $C()$ value among all legal tuples whose last coordinate is $\max(\{1, 2, \ldots, n\} \setminus S)$. By the discussion in Section 2.1 the given algorithm computes correctly the optimal total weighted tardiness. We examine now the running time. Let σ^* be an optimal sequence minimizing the total weighted tardiness. We use $T(\sigma^*)$ to denote the minimum total weighted tardiness of σ^* and $T_{\max}(\sigma^*)$ for the maximum tardiness of the jobs in σ^*.

Let Λ be the size of the state space. Given the state represented by $\mathcal{T}$ the computation time of the minimum cost for the state is upperbounded as follows: (i) In the Early Case the number of tuples to be examined is equal to the number of ways of distributing the p_j units of processing time into the $r \leq m$ bins corresponding to early intervals for the job. Taking into account the possibility that $p_j \leq r$, this quantity is at most $r!\binom{p_j+r}{r} \leq m!\binom{p_{\max}+m}{m}$, where $p_{\max} = \max_i p_i$. (ii) In the Tardy Case we need $O(m)$ time since there are $O(m)$ tuples $\mathcal{T}'$ to be examined. Therefore the time complexity of the dynamic programming algorithm will be $O(m!\binom{p_{\max}}{m}\Lambda)$. We give two bounds on Λ, one depending on P and the other on $T_{\max}(\sigma^*)$.

Given that at most all of the m straddlers may be Φ the number of ways to fill up the m straddler slots is at most $m!\binom{n+m}{m}$. Given the set S, let $1 \leq l \leq m$ be the number of straddlers that are different from Φ. The number of possible placements of the straddlers with respect to the due dates $D_1, D_2, \ldots, D_m$ is at most $\prod_{k_i \neq \Phi} p_{k_i} \leq (\sum_{k_i \neq \Phi} p_{k_i}/l)^l \leq (P/l)^l$. This quantity is maximized when $l = m$. Therefore the maximum contribution of the straddlers to the size of the state space is a multiplicative $m!\binom{n+m}{m}(P/m)^m$ factor. Similar considerations as in the proof of Theorem 2.1 show that a crude upper bound on the number of possible τ_i for a fixed i is $O((P/m)^m)$. The reason is that again we enumerate m-tuples whose coordinates sum to at most the length of I_i, which is less than P. Therefore the number of all different τ_i tuples, $i = 1, \ldots, m+1$, is at most $(P/m)^{O(m^2)}$.

Taking into account the last coordinate of each tuple $\mathcal{T}$, which corresponds to a job j we conclude that $\Lambda = O(m!\binom{n+m}{m}(P/m)^{O(m^2)}n)$. The following theorem has been shown.

Theorem 2.2 *There is an algorithm with complexity* $O\left(m!\binom{p_{\max}+m}{m}(m!\binom{n+m}{m}(P/m)^{O(m^2)}n)\right)$ *which computes the minimum total weighted tardiness for a problem with* m *distinct due dates.*

We now bound the complexity in terms of $T_{\max}(\sigma^*)$ assuming this quantity is known. If only an upper bound $T \geq T_{\max}(\sigma^*)$ is known, we can obviously use this in the ensuing calculations.

First, observe that the number of states which need to be examined in the Early Case is at most $O(\Lambda)$. Therefore we can rather crudely upperbound the running time by $O(\Lambda^2)$, thus eliminating the $p_{\max}$ factor. Recall that all the straddlers are tardy by definition. The main difference in the analysis above for Λ is that the sums previously upperbounded by P are in fact upperbounded by

$nT_{\max}(\sigma^*)$: (i) Given a straddler k_i there is a maximum of $T_{\max}(\sigma^*)$ possible completion times of the straddler past D_i. (ii) The total length of tardy jobs within an interval cannot exceed $nT_{\max}(\sigma^*)$. We have shown the following theorem:

Theorem 2.3 *Let $T_{\max}(\sigma^*)$ be the maximum tardiness of schedule σ^* which is optimal for $1|\ |\sum_j w_j T_j$. Let T be any known upper bound on $T_{\max}(\sigma^*)$. There is an algorithm with complexity $O\left(\left(m!\binom{n+m}{m}(nT/m)^{O(m^2)}n\right)^2\right)$ which computes the minimum total weighted tardiness for a problem with m distinct due dates.*

3 A fully polynomial time approximation scheme

Similarly to [10], we are going to scale and round down the processing times and scale down the due dates by a constant K, which is to be determined later. Accordingly, let us define $\overline{d}_j \doteq d_j/K$ and $\overline{p}_j \doteq \lfloor p_j/K \rfloor$ for $j = 1, 2, ..., n$. Assume that we apply the pseudopolynomial algorithm of Theorem 2.3 to this scaled down problem and let σ_A be the optimal sequence found by the algorithm. Let $\overline{T}_{\sigma_A(j)}$ be the tardiness of the jth job in this sequence with the scaled down data and let $T_{\sigma_A(j)}$ be the tardiness of the same job in σ_A with the original data. Then we clearly have $\overline{T}_{\sigma_A(j)} \leq T_{\sigma_A(j)}/K$ for $j = 1, 2, ..., n$. Furthermore, $\overline{T}_{\sigma_A} \doteq \sum_{j=1}^{n} w_{\sigma_A(j)} \overline{T}_{\sigma_A(j)} \leq T(\sigma^*)/K$ since σ_A is optimal for the scaled down data. Let T'_{σ_A} denote the total weighted tardiness of the sequence σ_A when we use processing times $p'_j \doteq K\overline{p}_j$ for each job j and the original due dates d_j. Note that $p'_j = K\overline{p}_j \leq p_j \leq K(\overline{p}_j + 1)$. If we define $T_{\sigma_A} \doteq \sum_{j=1}^{n} w_{\sigma_A(j)} T_{\sigma_A(j)}$, then we can write

$$K\overline{T}_{\sigma_A} \leq T(\sigma^*) \leq T_{\sigma_A} \leq \sum_{j=1}^{n} w_{\sigma_A(j)} \max\{K \sum_{i=1}^{j} (\overline{p}_{\sigma_A(i)} + 1) - d_{\sigma_A(j)}, 0\}$$
$$\leq T'_{\sigma_A} + w_{\max} K n(n+1)/2, \tag{1}$$

where $w_{\max} \doteq \max_{1 \leq j \leq n} w_j$.

Furthermore,

$$K\overline{T}_{\sigma_A} = K \sum_{j=1}^{n} w_{\sigma_A(j)} \max\{\sum_{i=1}^{j} \overline{p}_{\sigma_A(i)} - \overline{d}_{\sigma_A(j)}, 0\}$$
$$= \sum_{j=1}^{n} w_{\sigma_A(j)} \max\{\sum_{i=1}^{j} K\overline{p}_{\sigma_A(i)} - d_{\sigma_A(j)}, 0\} = T'_{\sigma_A} \tag{2}$$

Combining (1) and (2), we obtain

$$T'_{\sigma_A} \leq T(\sigma^*) \leq T_{\sigma_A} \leq T'_{\sigma_A} + w_{\max} K n(n+1)/2,$$

which implies

$$T_{\sigma_A} - T(\sigma^*) \leq w_{\max} K n(n+1)/2. \tag{3}$$

Since we do not need to consider schedules for which $T_{\sigma_A(j)}$ would exceed $T_{\max}(\sigma^*)$ for any $j \in \{1, 2, ..., n\}$, assuming the latter value was known, the complexity of the dynamic program described by Theorem 2.3 for the scaled problem will be bounded by

$$O\left(\left(m!\binom{n+m}{m}(nT_{\max}(\sigma^*)/Km)^{O(m^2)}n\right)^2\right)$$

It is well known that the earliest due date (EDD) order minimizes the maximum tardiness with any number of due dates [6]. Let $T_{\max}$ be the maximum tardiness and T_{EDD} the total weighted tardiness of this schedule. We can assume without loss of generality that $w_j \geq 1$ for all jobs j. Then we have

$$T_{\max} \leq T_{\max}(\sigma^*) \leq T(\sigma^*) \leq T_{EDD} \leq nw_{\max}T_{\max}. \tag{4}$$

Let us assume now that $w_{\max}$ does not grow too fast with n, i.e., there is a polynomial $g(n)$ such that we have $w_{\max} \leq g(n)$. If we choose $K = \varepsilon w_{\max}T_{\max}/(g^2(n) \cdot n(n+1)/2)$, then substituting into inequality (3) and using (4) yields

$$T_{\sigma_A} - T(\sigma^*) \leq g(n)Kn(n+1)/2 \leq \varepsilon T_{\max} \leq \varepsilon T_{\max}(\sigma^*) \leq \varepsilon T(\sigma^*).$$

Furthermore, the algorithm's complexity is upperbounded by

$$O\left(\left(m!\binom{n+m}{m}(n^2w_{\max}T_{\max}/(Km))^{O(m^2)}n\right)^2\right) =$$
$$O\left(\left(m!\binom{n+m}{m}(n^4g^2(n)/(\varepsilon m))^{O(m^2)}n\right)^2\right)$$

Thus we have proved the following.

Theorem 3.1 *If the job weights are bounded by a polynomial in n, then there is a fully polynomial time approximation scheme (FPTAS) for the minimum total weighted tardiness problem on a single machine with any fixed number of distinct due dates.*

4 Arbitrary number of due dates

4.1 An approximation bound obtained in quasipolynomial time

In this section we examine a general instance of the problem with an arbitrary number of due dates. Our goal is to transform the given instance into one with a reduced, although not necessarily constant, number of due dates. We then apply our previous algorithm whose complexity depends exponentially on the number of distinct due dates.

We are given an instance I with due dates d_j, $j = 1, \ldots, n$ and we will produce an instance I' with due dates d'_j, $j = 1, \ldots, n$. For a given schedule σ let $cost(\sigma)$ denote the total weighted tardiness under the d_j and $cost'(\sigma)$ the total weighted tardiness under the d'_j. Similarly use T_j, T'_j to denote the tardiness of job j in each case with reference to the same schedule σ. Let the *original*

optimum OPT refer to the optimal total weighted tardiness of instance I under the d_j and the *modified optimum* OPT' to the optimal total weighted tardiness of I' under the d'_j.

What is a good way to generate d'_j? Assume for example that we adopt the following strategy: for every job j, enforce $d'_j < d_j$. Then for a fixed schedule σ, we have $T'_j \geq T_j$, for all j, and hence $cost'(\sigma) \geq cost(\sigma)$. Computing σ as a near-optimal schedule for the d'_j forces us to shoot for a modified optimum $OPT' \geq OPT$. When we calculate the cost of σ under the original d_j it will potentially decrease. In order to analyze the performance guarantee we have to deal with two opposing effects: (i) upperbound the increase of OPT' with respect to OPT and (ii) lowerbound the difference $cost'(\sigma) - cost(\sigma)$. Symmetric considerations apply if we choose to set $d'_j > d_j$ for every j.

A mixed strategy where for some jobs the due dates increase and for others the due dates decrease seems to be more flexible. To counter the opposing effects inherent in the analysis, one could use randomization: for every job j we determine a_j, b_j such that $d_j \in [a_j, b_j]$, and we set d'_j to a_j with some probability λ_j and to b_j with probability $1 - \lambda_j$. We tried the randomized approach but could not get a bound better than the one we are about to present.

We choose for every j to set $d'_j := b_j$ for an appropriate value $b_j \geq d_j$. The b_j values will be determined at the end. It is clear, however, that $OPT' \leq OPT$ for any b_j values. We emphasize again that for time efficiency the resulting number of distinct due dates and hence the number of distinct b_j values must be small.We now relate the cost of a schedule under the original due dates d_j with the cost induced by the same schedule under the due dates d'_j. Later on, we will compute a near-optimal sequence with respect to the increased due dates and then we will have to argue about its cost with respect to OPT. The following lemma is straightforward.

Lemma 4.1 *For any sequence* σ, $cost(\sigma) \leq cost'(\sigma) + \sum_j w_j(b_j - d_j)$.

Observe that in the upcoming theorem we consider for added generality the existence of a non-standard approximation scheme that finds a $(1+\varepsilon)$-approximation for $\varepsilon \geq 0$, i.e., we also consider the existence of an exact algorithm.

Theorem 4.1 *Let* I' *be an instance derived from* I *based on the transformation defined above and let* $\mathcal{A}$ *be an approximation scheme for total weighted tardiness with running time* $T(\mathcal{A}, I', \varepsilon)$ *on instance* I' *for any* $\varepsilon \geq 0$. *We can compute in time* $T(\mathcal{A}, I', \varepsilon)$ *a schedule* σ *such that*

$$cost(\sigma) \leq (1+\varepsilon)OPT + \sum_j w_j(b_j - d_j).$$

Proof. We know that $OPT' \leq OPT$. Invoking algorithm $\mathcal{A}$ on I' yields a schedule σ with cost $cost'(\sigma) \leq (1+\varepsilon)OPT'$, which implies

$$cost'(\sigma) \leq (1+\varepsilon)OPT' \leq (1+\varepsilon)OPT.$$

Mapping back the due dates to the original d_j values yields by Lemma 4.1

$$cost(\sigma) \leq (1+\varepsilon)OPT' + \sum_j w_j(b_j - d_j) \leq (1+\varepsilon)OPT + \sum_j w_j(b_j - d_j).$$

■

We demonstrate now a way to define the b_j's. We follow the method of partitioning the time horizon from 0 to $\sum_j p_j$ into geometrically increasing intervals whose endpoints are powers of $1+\delta$ for fixed $\delta > 0$. Any due date that falls on a power of $1+\delta$ or at the endpoints of the time horizon is left unchanged. Otherwise if $d_j \in ((1+\delta)^l, (1+\delta)^{l+1})$ define $b_j := (1+\delta)^{l+1}$ and denote l by l_j. Observe that for different j, the l_j values may coincide. Let L denote the number of distinct due dates after this transformation. Under the assumption that the processing times are bounded by a polynomial in n, we can apply the algorithm described in Theorem 2.1 on the transformed instance I'. The running time of the algorithm will be $O(n^{O(L)})$. In our case $L = \lceil \log_{1+\delta} \sum_j p_j \rceil + 2$, therefore we obtain that $L = O(\log n / \log(1+\delta))$ under our assumption, i.e., the algorithm will be quasipolynomial.

Theorem 4.2 *If the job processing times are bounded by a polynomial in n, then for any fixed $\delta > 0$, we can compute in quasipolynomial time a schedule σ such that*

$$cost(\sigma) \leq OPT + \delta \sum_j w_j d_j.$$

Proof. Consider the instance I' produced from the original instance by the above transformation. By using Theorem 2.1, Theorem 4.1 applies with $\varepsilon = 0$ and one can compute a schedule σ such that

$$cost(\sigma) \leq OPT + \sum_j w_j \left((1+\delta)^{l_j+1} - d_j) \right) \tag{5}$$

We now upperbound the additive error term for job j.

$$w_j((1+\delta)^{l_j+1} - d_j) \leq w_j((1+\delta)^{l_j}(1+\delta-1)) =$$

$$\delta w_j (1+\delta)^{l_j} \leq \delta w_j d_j.$$

The theorem follows. ■

4.2 The relation with the total weighted late work objective

Since the optimal objective value for an instance of $1|\ |\Sigma_j w_j T_j$ can be zero, obtaining an approximation within a guaranteed multiplicative factor is difficult. In this section we describe a polynomial-time algorithm which guarantees a solution within a polynomial factor under the assumption that the total processing time does not grow faster than some polynomial $h(n)$. This is always worse than the $(n-1)$-approximation of [1]. We think it is worthwhile though to relate the performance of algorithms for $1|\ |\Sigma_j w_j V_j$ to the $1|\ |\Sigma_j w_j T_j$ problem.

Hariri et al. [5] have given an $O(n^2 P)$ pseudopolynomial-time algorithm and Kovalyov et al. [8] an FPTAS for $1|\ |\Sigma_j w_j V_j$. Naturally, when $P = O(h(n))$ for some polynomial $h(n)$ then $1|\ |\Sigma_j w_j V_j$ becomes polynomially solvable. In the following we show how this fact can be used for a polynomial-time approximation algorithm for $1|\ |\Sigma_j w_j T_j$.

Lemma 4.2 *For any schedule* σ

$$T(\sigma) = \sum_j w_{\sigma(j)} T_{\sigma(j)} \leq P \sum_j w_{\sigma(j)} V_{\sigma(j)} = PV(\sigma)$$

Proof. Let us define the lth *modified due date* for each job by $d_i^{(l)} \doteq d_i + lp_i$ for $i = 1, 2, ..., n$ and $l = 0, 1, 2, ...$ Consider an arbitrary schedule σ and let us denote by $T_{\sigma(j)}^{(l)}$ and $V_{\sigma(j)}^{(l)}$ the tardiness and the late work of job $\sigma(j)$ with respect to the due date $d_{\sigma(j)}^{(l)}$ for $j = 1, 2, ..., n$ and $l = 0, 1, 2, ...$ It is clear that for each l

$$T_{\sigma(j)}^{(l)} = V_{\sigma(j)}^{(l)} + T_{\sigma(j)}^{(l+1)} \tag{6}$$

and repeatedly applying (6) we obtain

$$T_{\sigma(j)} = V_{\sigma(j)} + T_{\sigma(j)}^{(1)} = V_{\sigma(j)} + V_{\sigma(j)}^{(1)} + T_{\sigma(j)}^{(2)} = ... = V_{\sigma(j)} + \sum_{k=1}^{l-1} V_{\sigma(j)}^{(k)} + T_{\sigma(j)}^{(l)} \tag{7}$$

Notice, however, that for any job i after at most $l_i = \lceil (P - d_i)/p_i \rceil$ iterations, we have $d_i^{(l_i)} \geq P$. This implies that $T_{\sigma(j)}^{(l_{\sigma(j)})} = V_{\sigma(j)}^{(l_{\sigma(j)})} = 0$ for job $\sigma(j)$. Furthermore, it is clear that

$$V_{\sigma(j)} \geq V_{\sigma(j)}^{(1)} \geq ... \geq V_{\sigma(j)}^{(l_{\sigma(j)})},$$

which yields after substitution into (7)

$$T_{\sigma(j)} \leq l_{\sigma(j)} V_{\sigma(j)} \leq P \cdot V_{\sigma(j)}. \tag{8}$$

Multiplying (8) by $w_{\sigma(j)}$ and summing over all j proves the bound of the lemma. ∎

It is easy to see that the bound of the lemma is asymptotically tight even for the unit weight case. Consider n identical jobs with unit processing time, n even. Let them have a common due date $n/2$. For any sequence the total late work is $n/2$ while the total tardiness is $(n/2)(n/2 + 1)$.

Theorem 4.3 *Let* σ_V *be an optimal schedule for* $1|\ |\Sigma_j w_j V_j$. *If the processing times do not grow too fast, i.e., there is a polynomial* $h(n)$ *such that* $P = O(h(n))$, *then* σ_V *can be obtained in polynomial time and the tardiness* $T(\sigma_V)$ *of this schedule is within a polynomial factor of the optimum for* $1|\ |\Sigma_j w_j T_j$, *i.e.,*

$$T(\sigma_V) \leq h(n) T(\sigma^*).$$

Proof. The theorem immediately follows by applying Lemma 4.2 to σ_V:

$$T(\sigma^*) \leq T(\sigma_V) \leq PV(\sigma_V) \leq PV(\sigma^*) \leq h(n) T(\sigma^*).$$

∎

5 Open questions

The obvious open question is how to improve the approximability of the problem with an arbitrary number of distinct due dates. We identify an additional open question. Is there an FPTAS for a fixed number of due dates, irrespective of the weight values?

Acknowledgement. The authors thank two anonymous referees for useful suggestions.

References

[1] T. C. E. Cheng, C. T. Ng, J. J. Yuan, and Z. H. Liu. Single machine scheduling to minimize total weighted tardiness. *European Journal of Operational Research*, 165:423–443, 2005.

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

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

[4] G. Hardy, J. E. Littlewood, and G. Pólya. *Inequalities, 2nd edition.* Cambridge University Press, 1952.

[5] A. M. A. Hariri, C. N. Potts, and L. N. Van Wassenhowe. Single machine scheduling to minimize total weighted late work. *ORSA J. on Computing*, 7:232–242, 1995.

[6] J. R. Jackson. Scheduling a production line to minimize maximum tardiness. Res. Rep. 43, Management Science Research Project, UCLA, 1955.

[7] S. G. Kolliopoulos and G. Steiner. On minimizing the total weighted tardiness on a single machine. In V. Diekert and M. Habib, editors, *Proceedings of the 21st Annual Int. Symposium on Theoretical Aspects of Computer Science*, volume 2996 of *Lecture Notes in Computer Science*, pages 176–186, Berlin, 2004. Springer-Verlag.

[8] M.Y. Kovalyov, C.N. Potts, and L.N. Van Wassenhove. A fully polynomial approximation scheme for scheduling a single machine to minimize total weighted late work. *Mathematics of Operations Research*, 19:86–93, 1994.

[9] E. L. Lawler. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, 1:331–342, 1977.

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

[11] E. L. Lawler and J. M. Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16:77–84, 1969.

[12] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[13] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6:1–12, 1959.

[14] C. N. Potts and L. N. Van Wassenhove. Single machine scheduling to minimize total late work. *Operations Research*, 40:586–595, 1992.

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