J Sched (2010) 13: 561–576
DOI 10.1007/s10951-010-0181-1

# Minimizing total weighted tardiness on a single machine with release dates and equal-length jobs

**J.M. van den Akker · G. Diepen · J.A. Hoogeveen**

Published online: 1 May 2010


**Abstract** In this paper we study the problem of scheduling $n$ jobs with release dates, due dates, weights, and equal processing times on a single machine. The objective is to minimize total weighted tardiness. We formulate the problem as a time-indexed ILP after which we solve the LP-relaxation. We show that for certain special cases (namely when either all due dates, all weights, or all release dates are equal, or when all due dates and release dates are equally ordered), the solution for the LP-relaxation is either integral or can be adjusted in polynomial time into an integral one. For the general case we present a branching rule that performs well. Furthermore we show that the same approach holds for the $m$ identical, parallel machines variant of the problem. Finally we show that with a minor modification the same approach also holds for the single-machine problems of minimizing the sum of weighted late jobs $(1|r_j, p_j = p|\sum w_j U_j)$ and the sum of weighted late work $(1|r_j, p_j = p|\sum w_j V_j)$ as well as their respective variants with $m$ identical, parallel machines. We further show how we can solve these problems by applying column generation when there is not sufficient memory available to apply the direct ILP-approach.

**Keywords** Common processing time · Weighted tardiness · Time indexed formulation

## 1 Introduction

The problem we are looking at is the following: We have a single machine on which we have to schedule a set $N = \{1, 2, \ldots, n\}$ of jobs, where $n$ is the number of jobs. For each job $J_j$ we have a release date $r_j$ before which job $J_j$ is not available, a due date $d_j$, and a weight $w_j$. The processing times of the jobs are all equal to $p$. We assume the due dates, the release dates, the weights, and the common processing time of all jobs to be integral. We are looking for a feasible schedule, that is, we want to find a set of completion times $C_j$ $(j = 1, \ldots, n)$ such that no job starts before its release date and no two jobs overlap in their execution. Given the completion time $C_j$ of a job $J_j$, we define the tardiness $T_j$ of job $J_j$ as:

$$T_j = \max\{0, C_j - d_j\}.$$

Now the objective is to find the feasible schedule that minimizes the total weighted tardiness. To the best of our knowledge the computational complexity of this problem is still open.

For writing down the different problems we make use of the three-field notation scheme introduced by Graham et al. (1979). In this three-field notation scheme the problem of minimizing total weighted tardiness on a single machine with release dates and equal-length jobs is denoted as $1|r_j, p_j = p|\sum w_j T_j$.

Supported by BSIK grant 03018 (BRICKS: Basic Research in Informatics for Creating the Knowledge Society).

J.M. van den Akker · J.A. Hoogeveen
Department of Computer Science, Utrecht University, P.O. Box 80089, 3508 TB Utrecht, The Netherlands

J.M. van den Akker
e-mail: marjan@cs.uu.nl

J.A. Hoogeveen
e-mail: slam@cs.uu.nl

G. Diepen (✉)
Paragon Decision Technology, Schipholweg 1, 2034 LS Haarlem, The Netherlands
e-mail: Guido.Diepen@aimms.com

Over the years quite some research has been done on scheduling problems regarding (weighted) tardiness. Lawler (1977) gave a pseudopolynomial algorithm for solving the $1 \parallel T_j$ problem and Du and Leung (1990) gave a proof for the $1 \parallel T_j$ problem to be $\mathcal{NP}$-hard in the ordinary sense. The weighted version of this problem, $1 \parallel w_j T_j$, is known to be $\mathcal{NP}$-hard in the strong sense (Lenstra et al. 1977). Akturk and Ozdemir (2001) gave a new dominance rule for solving the $1|r_j|w_j T_j$ problem to optimality using branch-and-bound.

Also quite some research has been done on scheduling problems with equal processing times: Baptiste (2000) looks at the $1|r_j, p_j = p|\sum T_j$ problem, as well as the problem with $m$ identical, parallel machines instead of one and shows that both problems can be solved in polynomial time by means of dynamic programming. Baptiste (1999) gives a polynomial time algorithm for the $1|r_j, p_j = p|\sum w_j U_j$ problem based on dynamic programming. In Baptiste et al. (2004) ten equal-processing-time scheduling problems are shown to be solvable in polynomial time, among which is the $Pm|r_j, p_j = p|\sum w_j U_j$ problem. An overview of more problems with equal processing times can be found in Leung (2004).

Verma and Dessouky (1998) look at common processing time scheduling with earliness and tardiness penalties. They formulate this problem as a time-indexed ILP and show that when certain criteria are met, there exists an integral optimal solution to the LP-relaxation, which means that there exists a polynomial time solution procedure. In this paper we will follow their approach, and we will show that if certain criteria hold, then the $1|r_j, p_j = p|\sum w_j T_j$ problem can be solved in polynomial time.

The outline for the rest of this paper is as follows: In Sect. 2 we give an ILP-formulation of the problem. In Sect. 3 we will present an algorithm that can be used to rewrite fractional solutions for the single-machine problem that possess a special condition of being non-double nested. In Sect. 4 we will discuss the special case of the problem in which the jobs have a common due date, and in Sect. 5 we will discuss the general problem. In Sect. 6 we will apply the same techniques on problems with related objective functions for the single-machine case, and in Sect. 7 we look at the case with $m$ parallel, identical machines. After that in Sect. 8 we will discuss another way of solving the problem, which aims at reducing the amount of memory needed, and in Sect. 9 we will present some experimental results. Finally in Sect. 10 we will draw some conclusions.

## 2 Problem formulation

Like in Verma and Dessouky (1998), we use a time-indexed formulation to represent the problem as an ILP. We restrict ourselves to those times that can occur as completion times in an optimal solution. Because of the equal processing times, the processing of a job will always occur in an interval with length $p$. We denote each interval by the end time of the interval. The objective function that we consider is total weighted tardiness, and since this is a regular function, there will always exists an optimal schedule that is left-aligned, that is, it is not possible to execute any job earlier without postponing any other job.

Since there exists an optimal schedule that is left-aligned, we can restrict ourselves to schedules in which each job, either starts at its release date, or immediately after another job. Hence, each release date introduces a set of possible completions times, and in the extreme case some job $J_j$ will start right at its release date after which all other jobs follow contiguously. This means that job $J_j$ introduces $(n-1)$ possible completion times for the other jobs.

Unless another job $J_{j'}$ has a release date that is a multiple of $p$ before $r_j$, only job $J_j$ can be completed at time $\gamma_j = r_j + p$. We define the set $G$ denoting all first possible completion times of all jobs by

$$G := \bigcup_{j=1}^{n} \{\gamma_j\}.$$

We define the set $M_j$ as the set of possible completion times introduced by job $J_j$ as

$$M_j = \{r_j + 2p, r_j + 3p, \ldots, r_j + np\},$$

and the set $K$ containing all other possible completion times as

$$K := \bigcup_{j=1}^{n} M_j.$$

Since the first possible completion time of job $J_j$ is $\gamma_j$, we find that the set of possible completion times of a job $J_j$ is

$$A_j := \{t \in K | t > r_j + p\} \cup \{\gamma_j\}.$$

Since the capacity of the machine is equal to 1, completing a job $J_j$ at time $t$ implies that no other job $J_{j'}$ can have a completion time that is fewer than $p$ time units before $t$. For a given time $t$, we define $H_t$ as the set of preceding completion times conflicting with $t$ as

$$H_t := \left\{t' \in K \cup G \middle| 0 \le t - t' < p\right\}.$$

Now we formulate the problem as a time-indexed ILP model. For the ILP-formulation we define a variable $x_{j,t}$ for all relevant $j$ and $t$ as

$$x_{j,t} = \begin{cases} 1 & \text{if job } J_j \text{ is completed at time } t, \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore we define the cost $c_{j,t}$ of having job $J_j$ being completed at time $t$ as

$$c_{j,t} = w_j \max\{0, t - d_j\}.$$

Now the complete ILP model becomes

$$\min z = \sum_{j=1}^{n} \sum_{t \in A_j} c_{j,t} x_{j,t}$$

subject to

$$\sum_{t \in A_j} x_{j,t} = 1 \quad \text{for all } j \in N, \tag{1}$$

$$\sum_{j=1}^{n} \sum_{t' \in H_t \cap A_j} x_{j,t'} \leq 1 \quad \text{for all } t \in K \cup G, \tag{2}$$

$$x_{j,t} \in \{0, 1\} \quad \text{for all } j \in N, t \in A_j, \tag{3}$$

where constraint (1) ensures that all jobs are assigned to exactly one interval and constraint (2) ensures that for any time interval no more than one job is processed.

Verma and Dessouky (1998) look at the single-machine scheduling of equal-length jobs with both earliness and tardiness penalties, where earliness for job $J_j$ is defined as $E_j = \max\{0, d_j - C_j\}$. In the three-field notation scheme this problem is $1|p_j = p|\sum \alpha_j E_j + \beta_j T_j$, where $\alpha_j$ is the earliness penalty for job $J_j$ and $\beta_j$ is the tardiness penalty for job $J_j$. Since they consider earliness penalties, sometimes it can be beneficial to introduce idle time before starting to process a job. They use the same time-indexed formulation. Except for the objective function, which is reflected in the values $c_{jt}$, the only difference with our problem is the presence of release dates. We build on their results. First, we repeat some of their definitions.

**Definition 1** (Verma and Dessouky 1998) Let $x$ be a feasible solution to the LP-relaxation. We say that job $J_{j_2}$ is *nested* in job $J_{j_1}$ ($j_1 \neq j_2$), if there exist values $t_k \in K \cup G$ ($k = 1, 2, 3$) such that $t_1 < t_2 < t_3$ and $x_{j_1,t_1}$, $x_{j_2,t_2}$ and $x_{j_1,t_3}$ are all positive.

**Definition 2** (Verma and Dessouky 1998) A feasible solution of the LP-relaxation is double nested if and only if there exists a pair of jobs $J_{j_1}$ and $J_{j_2}$ which are nested in each other. If no such pair of two jobs exists, then the solution is called nested or non-double nested.

Verma and Dessouky (1998) use the term non-nested solution instead of double-nested solution. For reasons of clarity we changed this term into double nested.

For ease of writing we furthermore define the notation 1212 to denote that there exist $x_{j_1,t_1}$, $x_{j_2,t_2}$, $x_{j_1,t_3}$, and $x_{j_2,t_4}$ with $t_1 < t_2 < t_3 < t_4$ all having value $> 0$ in the solution of the LP.

Verma and Dessouky (1998) show that the $1|p_j = p|\sum \alpha_j E_j + \beta_j T_j$ problem is polynomially solvable if the jobs can be indexed such that $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_n$ and $\beta_1 \leq \beta_2 \leq \cdots \leq \beta_n$. To show this, they prove two results. First, they prove that there exists an optimal solution to the LP-relaxation that is non-double nested, and that in case of a strict ordering, in which each inequality is strict, each optimal solution is non-double nested. Second, they prove that all extremal, non-double-nested solutions to the LP-relaxation are integral. This settles the complexity of the problem with the strict ordering; for the case with equal weights they show how these equalities can be removed by adding perturbations such that the resulting solution is still optimal for the original problem.

The proof that each non-double-nested, extremal solution of the LP-relaxation is integral is independent from the objective function, as it is solely based on the fact that if the solution is nested, then the constraint matrix for the non-zero columns (i.e. columns for which the value is greater than 0) of the solution can be reordered in such a way that it forms an interval matrix, which are known to be totally unimodular (Nemhauser and Wolsey 1988). Therefore, we can use their results, if we show that the presence of the release dates does not destroy their proofs. Since a release date $r_j$ of job $J_j$ implies that $J_j$ cannot complete its execution at a time $t < r_j + p$, we only have to add a set of constraints of the form $x_{jt} = 0$ for these values of $t$. These constraints are represented by unit rows in the constraint matrix and adding a unit row to a totally unimodular matrix results in a matrix which is also totally unimodular. Hence, we get the following lemma.

**Lemma 1** *The result of Verma and Dessouky (1998) concerning the integrality of any nested extremal solution are not influenced by adding release dates.*

Because of the result by Verma and Dessouky (1998) that any nested extremal solution is integral, we can show that a problem is solvable in polynomial time by showing that any double-nested solution to the LP-relaxation is sub-optimal. If this is not possible, then we must show that an optimal double-nested solution can be converted into a non-double-nested solution with equal cost, which then can be converted into an integral solution with equal cost. Remark that perturbing the objective function can be done independently from the addition of release dates as well; we will describe this technique in the next section.

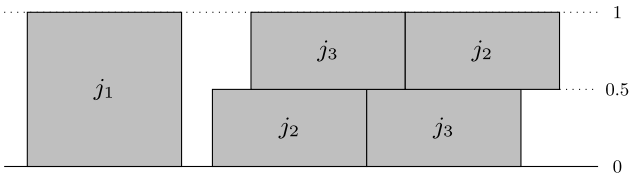


**Fig. 1** Example of a fractional solution in the LP-relaxation

One important observation that has to be made is that allowing fractional solutions in the LP-relaxation is not the same as allowing preemption. With preemption you are allowed to process some part of the job and after any time that is smaller than the processing time, you may preempt it. In our case, if jobs have fractional assignments, no preemption occurs: each job will always have the same processing time, only a certain fraction of the work needed for the job is processed in that time interval. An example of this situation is given in Fig. 1. The maximum capacity of the machine is equal to 1 (i.e. at most one job can be processed at the same time). Job $j_1$ has an integer allocation value, and therefore it is completely processed in one given time interval, whereas both jobs $J_{j_2}$ and $J_{j_3}$ each have fractional allocation values, and hence they are both partially allocated to two different time intervals.

## 3 Converting fractional non-double-nested solutions

In this section we present two algorithms to convert a fractional solution that is non-double nested into an integral solution with equal cost. The first one is a quick heuristic, the success of which is not guaranteed; the second one is based on the perturbation technique by Verma and Dessouky (1998). We start with the heuristic. Given a fractional, non-double-nested solution, we modify it, using a straightforward preprocessing routine, such that

- The assignment is left-aligned, that is, given that job $J_j$ is completed at time $t$, it is not possible to increase any value $x_{j,t'}$ with $r_j + p \leq t' < t$, while keeping the assignment values of the other jobs intact.
- There are no two jobs $J_i$ and $J_j$ that are both (partly) assigned to the same pair of intervals.

We use the following algorithm to look for a feasible, optimal, integral solution. Here we distinguish between *initially full* intervals and *initially idle* interval. An interval $[t, t + p]$ is called initially full, if the machine is working at full capacity during the sub-interval $[t, t + \delta]$, where $\delta$ is some small positive value, and it is called initially idle, otherwise. In Fig. 1 for example, the first interval to which job $J_{j_2}$ has been assigned is initially idle, whereas the second interval to which job $J_{j_2}$ has been assigned is initially full.

SELECTION ALGORITHM

- Start with the leftmost interval.
- If this interval is initially full, then pick a job assigned to this interval in the following way:
  - If there is only one job, pick this one.
  - If there are two or more jobs assigned to this interval and each one has been picked before, pick any of them.
  - Otherwise, pick a job that has not been picked before; if there are two (or more) such jobs, pick any job that has been assigned to some earlier interval.
  - Record the job-interval combination and move to the interval that starts $p$ time units later, unless this one was the last interval.
- If the interval is initially idle, then pick any job that has not been picked before; if all jobs have been picked before, then do not pick any job.
- Record the job-interval combination and move to the next interval, unless this one was the last interval.

**Theorem 1** *If each job occurs in exactly one recorded job-interval combination, then the schedule that is obtained by assigning each job to the interval corresponding to the selected job-interval combination is feasible and optimal.*

*Proof* We start by showing feasibility. Because of the Selection Algorithm, the jobs are assigned to non-overlapping intervals. Moreover, since each of the job to interval assignments occurs in the fractional solution as well, we do not violate any release dates. As each job is assigned exactly once, the corresponding schedule is feasible.

Let $Z$, with corresponding values $z_{j,t}$ for all relevant combinations of $j$ and $t$, denote the assignment obtained by the Selection Algorithm. Furthermore, let $X$ with values $x_{j,t}$ denote the assignment found by solving the LP-relaxation. Now we construct the assignment $Y$ with values $y_{j,t} = x_{j,t} - \epsilon z_{j,t}$ for all relevant combinations of $j$ and $t$, where $\epsilon$ is equal to some small value that we determine according to the following two constraints. First of all, $\epsilon$ has to be no more than the smallest positive assignment value in $X$. Because of this constraint, and since $z_{j,t} = 1$ can occur only if $x_{j,t} > 0$, we see that all $y_{j,t} \geq 0$. Moreover, $\epsilon$ should be no more than the minimum amount of spare machine capacity that is available in the sub-interval $[t, t + \delta]$ of any initially idle interval $[t, t + p]$ from which no job was picked. Because of this constraint, and since the Selection Algorithm always selects a job whenever the machine is working at full capacity, the machine never requires more capacity than $1 - \epsilon$ to execute assignment $Y$. Furthermore, the total amount of job $J_j$ $(j = 1, \ldots, n)$ that has been assigned in $Y$ is equal to $1 - \epsilon$.

Hence, if we multiply each $y_{j,t}$ value with a factor $1/(1 - \epsilon)$, then we get another feasible solution, $Y'$, to the

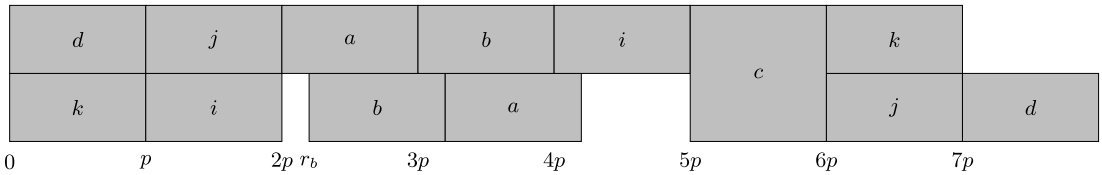


**Fig. 2** Bad example for the Selection Algorithm

LP-relaxation. This implies that $X$ can be written as a convex combination of $Y'$ and $Z$; because of the optimality of $X$, assignment $Z$ must be optimal, too. □

Unfortunately, the Selection Algorithm does not always produce a feasible schedule. Consider the following example (Fig. 2).

All jobs have release date 0, except for jobs $J_b$ and $J_c$, the release dates of which coincide with the start points of the first intervals they have been assigned to.

If we apply the Selection Algorithm, then we must select one of the jobs $J_d$ and $J_k$ at time 0, and one of the jobs $J_i$ and $J_j$ at time $p$. Suppose that we have chosen to select jobs $J_d$ and $J_j$. At time $2p$ we select job $J_a$, since this is a yet unselected job in a partially idle interval. We continue with $J_b$ at time $3p$, job $J_i$ at time $4p$, job $J_c$ at time $5p$, job $J_k$ at time $6p$. At time $7p$, we do not select any job, since the interval is partially idle and job $J_d$ has already been selected. This assignment corresponds to a feasible schedule that must be optimal according to Theorem 1.

If we alter the initial selection at times 0 and $p$ to jobs $J_d$ and $J_i$, then we again select jobs $J_a$ and $J_b$ at times $2p$ and $3p$, but at time $4p$ we are forced to select job $J_i$ again, which makes the Selection Algorithm to fail. We might change the Selection Algorithm such that no job is picked at time $2p$ (the interval is partially idle, and job $J_a$ must appear later again, since it has not been fully assigned yet), pick $J_b$ at time $r_b$, pick job $J_a$ at time $r_b + p$, wait until time $5p$ to pick job $J_c$, but then we fail at time $6p$, since we cannot pick jobs $J_j$ and $J_k$ both, which implies that one of these jobs will not be selected at all.

Fortunately, Verma and Dessouky (1998) have described a guaranteed way to convert a fractional solution that is non-double nested into an integral solution with equal cost. Since each extremal solution is integral, a fractional optimal solution must be a convex combination of two or more extremal solutions with equal cost. To exclude the possibility that two extremal solutions have equal cost, Verma and Dessouky (1998) add a small perturbation to the coefficients in the objective function; this perturbation is so small that an extremal solution to the original problem that is sub-optimal cannot become optimal for the perturbed problem. Adding perturbations is done in the following way:

$$\overline{c_{j,t}} = c_{j,t} + j\varepsilon$$

where $\varepsilon$ is a very small positive number.

## 4 Common due date

In this section we look at the special case in which all jobs $j$ have a common due date $d_j = D$. Depending upon the size of $D$, we distinguish two variants. First, in Sect. 4.1 we look at the case $0 \leq D < r_{\min} + p$, where $r_{\min}$ is the smallest release date of all jobs. For any value of $D < r_{\min} + p$ we know that each job will be tardy, which implies that we obtain an equivalent problem by putting $D = 0$. The $1|r_j, p_j = p, d_j = 0|\sum w_j T_j$ problem is equivalent to the $1|r_j, p_j = p|\sum w_j C_j$ problem, for which Baptiste (2000) already showed that it can be solved in polynomial time ($\mathcal{O}(n^7)$) by means of dynamic programming. In Sect. 4.2 we look at the case where $D \geq r_{\min} + p$; in this case it is possible to complete at least one job at or before the due date. For both cases we assume without loss of generality that the jobs are re-indexed such that $w_1 \leq w_2 \leq w_3 \leq \cdots \leq w_n$. If after re-indexing the jobs $w_1 < w_2 < w_3 < \cdots < w_n$ holds, then we say that a strict order on the weight of the jobs exists.

### 4.1 Common due date equal to zero

**Lemma 2** *If there exists a* strict *order on the weight of the jobs, then any optimal solution is nested. When no strict order exists, then there exists an optimal schedule that is nested.*

*Proof* Assume we have an optimal solution that is double nested. Therefore, according to Definition 2 there exist jobs $J_{j_1}$ and $J_{j_2}$ that are nested in each other. Without loss of generality we assume $j_1 < j_2$ (i.e. $w_{j_2} \geq w_{j_1}$). Let us now look at the situation where job $J_{j_1}$ is nested in job $J_{j_2}$, that is, there exist intervals $t_1$, $t_2$, and $t_3$ such that $t_1 < t_2 < t_3$ and $x_{j_2,t_1}$, $x_{j_1,t_2}$, and $x_{j_2,t_3}$ are all positive (i.e. situation 212). We define $\varepsilon = \min\{x_{j_1,t_2}, x_{j_2,t_3}\}$.

In Fig. 3 an example of the situation 212 is depicted where without loss of generality we assumed $\varepsilon = x_{j_2,t_3}$. Job $J_{j_1}$ is nested in job $J_{j_2}$, and job $J_{j_2}$ is nested in job $J_{j_1}$.

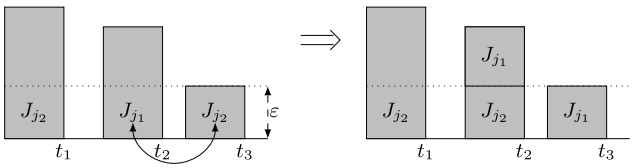


**Fig. 3** Example of rearranging $\varepsilon$ between two nested jobs.

Since both jobs are tardy, the exchange of $\varepsilon$ between $x_{j_1,t_2}$ and $x_{j_2,t_3}$ causes a change in the cost of $\Delta$, where we have:

$$\begin{aligned}\Delta &= \varepsilon(w_{j_2}t_2 + w_{j_1}t_3 - w_{j_1}t_2 - w_{j_2}t_3)\\ &= \varepsilon\big(w_{j_2}(t_2 - t_3) + w_{j_1}(t_3 - t_2)\big)\\ &= \varepsilon\big((t_3 - t_2)(w_{j_1} - w_{j_2})\big)\\ &\leq 0.\end{aligned}$$

In the resulting situation job $J_{j_2}$ is still nested in job $J_{j_1}$, but job $J_{j_1}$ is not nested in job $J_{j_2}$ anymore. The exchange does not violate any constraint as the amount of total allocation at each interval is unaltered, and only its division between the two jobs has been changed. Furthermore, the value of the objective function will not increase by this exchange. In case a strict order exists it can be seen that $\Delta < 0$, which contradicts the optimality of the double-nested solution.

If there is no strict order on the weight of the jobs, then we may have that $\Delta = 0$ for each of these interchanges. Hence, there can exist an optimal schedule that is double nested, but we can convert it into a nested schedule with equal cost by repeatedly moving parts of jobs with equal weight, like we did above. □

We have now shown that for the problem $1|r_j, p_j = p|\sum w_j C_j$ a double-nested solution is either optimal (but can be rewritten into a nested solution with equal cost in polynomial time) or sub-optimal. Now we can make use of Lemma 1 to show that this problem can be solved in polynomial time since the constraint matrix for the non-zero columns is totally unimodular. If the LP-solver returns a double-nested solution, then we first convert it into a nested solution as described in the proof of Lemma 2 and, if this nested solution is fractional, then we apply the Selection Algorithm of Sect. 3 to find a feasible schedule with equal cost. If the Selection Algorithm fails, then we can apply the perturbation technique by Verma and Dessouky.

Solving an LP with $n'$ variables takes $\mathcal{O}(\sqrt{n'}\log\frac{n'}{\epsilon})$ iterations of each $\mathcal{O}(n'^3)$ calculation steps (Roos 2005). In our case with $n$ jobs, the number of variables we have is $\mathcal{O}(n^3)$, which means that the total running time for the LP is $\mathcal{O}(n^{10}\sqrt{n}\log\frac{n^3}{\epsilon})$. Although in the worst-case this is more than the previous known result of Baptiste (2000) with dynamic programming, on average it might be a lot quicker.

### 4.2 Common due date ($d_j = D > p$)

Now we have the situation that there exists a common due date for all jobs that is bigger than the first possible completion time of any job. Hence, one or more jobs can be placed before the common due date while others will become tardy.

**Lemma 3** *A double-nested solution is either sub-optimal, or it can be rewritten into a nested solution with equal cost.*

*Proof* Assume that we have an optimal solution that is double nested. Therefore, there are two jobs $J_{j_1}$ and $J_{j_2}$ that are nested in each other; without loss of generality, we assume that $j_1 < j_2$. We address the situation that $J_{j_1}$ is nested in $J_{j_2}$. Then there exist time intervals $t_1$, $t_2$, and $t_3$ such that $t_1 < t_2 < t_3$ and $x_{j_2,t_1}$, $x_{j_1,t_2}$, and $x_{j_2,t_3}$ are positive.

Now there are two distinctions to be made:

- $D \geq t_3$. In this case the order of the jobs $J_{j_1}$ and $J_{j_2}$ does not matter because they are both on time. Hence it is possible to switch parts of the jobs around in such a way that we end up with a nested solution of equal cost.
- $D < t_3$. If the weights of the two jobs are equal, we can exchange parts between intervals $t_2$ and $t_3$ at no cost to convert the current double-nested solution into a nested solution. Otherwise we use the same approach as in the proof of Lemma 2, and we define $\varepsilon = \min(x_{j_1,t_2}, x_{j_2,t_3})$.

  The optimal way of allocating $\varepsilon$ of job $J_{j_2}$ and $\varepsilon$ of job $J_{j_1}$ among the two intervals is to first allocate $\varepsilon$ of job $J_{j_2}$ to interval $t_2$ and then allocate $\varepsilon$ of job $J_{j_1}$ to interval $t_3$. The idea behind this can be seen again in Fig. 3. If there exists a strict order on the weights of the jobs, the resulting solution after the exchange will have smaller cost. This implies that we can improve or rewrite the double-nested solution without violating any constraints as the amount of total allocation at each interval in unaltered, and only its division between the two jobs has been changed. Furthermore, no job is moved to an interval that is before any interval it was previously allocated to, and hence each release date is satisfied.

This process might have to be repeated if more occurrences exist of double-nested situations for these two jobs.

We see that either the double-nested solution is not optimal, which contradicts our assumption, or it can be rewritten into a nested solution of equal cost. □

For the case of equal due dates $D$ with any value for $D$ we present the following theorem:

**Theorem 2** *The problem* $1|r_j, p_j = p, d_j = D|\sum w_j T_j$ *can be solved in polynomial time.*

*Proof* By combining Lemmas 2 and 3 we know that for any value of $D$ there exists an optimal schedule that is nested.

We can then apply the Selection Algorithm to try to convert this nested, fractional solution to an integral solution with equal cost; if this is unsuccessful, then we apply the perturbation technique by Verma and Dessouky (1998). □

## 5 Arbitrary due dates

Now we look at the situation where all jobs can have different due dates. In this case we cannot provide a polynomial algorithm for solving the problem, but we give a good branching rule that makes use of the structure of the problem.

Unlike the common due date case, with arbitrary due dates a fractional solution of the LP-relaxation can have a better value than the optimal integer solution. An example that shows this is Table 1.

The optimal LP solution with cost 3 is shown in Fig. 4. The possible integral solutions are:

- start with job $J_1$. After that process job $J_3$ and job $J_4$ and finally process job $J_2$, which will be tardy. This solution has cost 5.
- start with job $J_2$. After that one of the jobs $J_1$, $J_3$, and $J_4$ will become tardy and due to the large weight of these jobs, the cost of this solution will be greater than 5.
- start with job $J_3$ or $J_4$. Since there is enough of free capacity to put job $J_1$ in front without causing a delay, this is dominated by the solution in which job $J_1$ is put first.

Thus the optimal ILP solution value equals 5, which is different from the optimal LP solution with value 3. Furthermore we can see in Fig. 4 that the jobs $J_1$ and $J_2$ are double nested.

In the following, we will analyze when it can be optimal for two jobs $J_{j_1}$ and $J_{j_2}$ to be double nested; we will further derive conditions under which we can show that such

**Table 1** Common processing time $p = 2$

| Job | Release date | Due date | Weight |
|---|---|---|---|
| 1 | 0 | 8 | 100 |
| 2 | 1 | 3 | 1 |
| 3 | 2 | 5 | 100 |
| 4 | 4 | 7 | 100 |

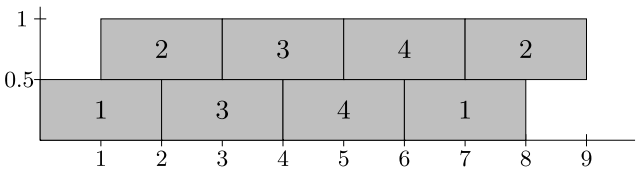


**Fig. 4** Optimal LP solution that is double nested

a double-nested situation is sub-optimal. We number the jobs such that $d_{j_1} \leq d_{j_2}$. Anticipating our results, we remark that in case of equal due dates a double-nested solution can be converted into a nested solution. There are two possible double-nested situations for the two jobs:

- Situation 1212
- Situation 2121

In both cases, we assume that the jobs have been partly assigned to intervals $t_1, t_2, t_3, t_4$ with $t_1 < t_2 < t_3 < t_4$.

**Lemma 4** *In the* 1212 *situation where there are two jobs* $J_{j_1}$ *and* $J_{j_2}$ *with* $d_{j_1} \leq d_{j_2}$, *a double-nested solution is either sub-optimal, or it can be rewritten into a nested solution with equal cost.*

*Proof* Assume $d_{j_2} \geq t_3$. In this case job $J_{j_2}$ will be on time when completed at time $t_3$. Exchanging $\varepsilon = \min\{x_{j_2,t_2}, x_{j_1,t_3}\}$ between $x_{j_2,t_2}$ and $x_{j_1,t_3}$ is profitable or yields the same cost, since:

- Job $J_{j_2}$ will still be on time.
- Job $J_{j_1}$ is placed earlier in time and thus its cost can only decrease or remain the same.

Now assume $d_{j_2} < t_3$. In this case both jobs are tardy at times $t_3$ and $t_4$. We have to make a distinction into two separate cases, based on the weight of the jobs:

- $w_{j_1} \geq w_{j_2}$. A straightforward calculation shows that, because $d_{j_1} \leq d_{j_2}$ and $w_{j_1} \geq w_{j_2}$, the value of the objective function will not increase when exchanging $\epsilon = \min\{x_{j_2,t_2}, x_{j_1,t_3}\}$ between $x_{j_2,t_2}$ and $x_{j_1,t_3}$.
- $w_{j_1} < w_{j_2}$. Since both jobs are tardy when completed at time $t_3$, exchanging $\varepsilon = \min\{x_{j_1,t_3}, x_{j_2,t_4}\}$ between $x_{j_1,t_3}$ and $x_{j_2,t_4}$ will increase the cost for job $J_{j_1}$ by $\epsilon w_{j_1}(t_4 - t_3)$ and decrease the cost for job $J_{j_2}$ by $\epsilon w_{j_2}(t_4 - t_3)$. Since $w_{j_1} < w_{j_2}$, the total cost decreases.

Hence, when $d_{j_1} \leq d_{j_2}$, the situation 1212 can always be rewritten to a nested solution of equal or less cost. □

In the case of $d_{j_1} = d_{j_2}$ we can choose the situation arbitrarily and thus we can always see it as the 1212-situation. This means that the case of $d_{j_1} = d_{j_2}$ can always be rewritten to a nested solution of equal or smaller cost.

**Lemma 5** *In the* 2121 *situation where there are two jobs* $J_{j_1}$ *and* $J_{j_2}$ *with* $d_{j_1} < d_{j_2}$, *a double-nested solution is either always sub-optimal, or it can be rewritten into a nested solution of the same cost, unless* $w_{j_1} < w_{j_2}$ *and* $r_{j_1} > r_{j_2}$.

*Proof* When $w_{j_1} = w_{j_2}$, we can always transform the current solution into a new solution of equal or smaller cost by exchanging between the intervals $t_3$ and $t_4$, since $d_{j_1} < d_{j_2}$.

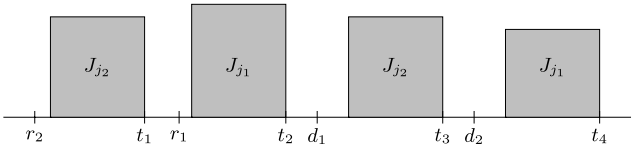


**Fig. 5** Example of situation where rewriting is not profitable

We now have to check the cases where the weights of the jobs are not equal.

Assume $w_{j_1} > w_{j_2}$. Now there are two options:

- $d_{j_2} \geq t_2$. Because $w_{j_1} > w_{j_2}$ and $d_{j_1} < d_{j_2}$ it will be profitable to exchange a bit of the last two parts of the jobs: $x_{j_2,t_3}$ and $x_{j_1,t_4}$. This exchange will never cause an increase in cost.
- $d_{j_2} < t_2$. In this case both jobs are tardy at times $t_3$ and $t_4$. Because $w_{j_1} > w_{j_2}$ it will always be profitable to exchange between $x_{j_2,t_3}$ and $x_{j_1,t_4}$.

Now assume $w_{j_1} < w_{j_2}$. Again there are two options:

- $d_{j_2} < t_2$. In this case both jobs are tardy at times $t_2$, $t_3$, and $t_4$. It will always be profitable to exchange $x_{j_1,t_2}$ and $x_{j_2,t_3}$, because you want the job with the larger weight ($J_{j_2}$ in this case) more to the front.
- $d_{j_2} \geq t_2$. In this case job $J_{j_2}$ will be on time at time $t_2$. Generally speaking we would like to exchange $x_{j_2,t_1}$ and $x_{j_1,t_2}$ because job $J_{j_2}$ is still on time at time $t_2$ and job $J_{j_1}$ is put earlier which is always at least as profitable. This is only guaranteed to be possible when $r_{j_1} \leq t_1 - p$ holds. This condition holds for sure when $r_{j_1} \leq r_{j_2}$.

The only case when the situation 2121 cannot always be improved or changed for free into a nested solution is when $d_{j_1} < d_{j_2}$, $w_{j_1} < w_{j_2}$ and $r_{j_1} > r_{j_2}$. An example of such a situation is depicted in Fig. 5. If $w_{j_2}$ is greater than or equal to $\frac{(t_4-t_3)w_{j_1}}{(t_4-d_{j_2})}$, then an exchange between the two intervals will not be profitable. □

**Theorem 3** *When there are no two jobs $J_{j_1}$ and $J_{j_2}$ with $r_{j_1} > r_{j_2}$, $d_{j_1} < d_{j_2}$ and $w_{j_1} < w_{j_2}$, then the problem $1|r_j, p_j = p|\sum w_j T_j$ can be solved in polynomial time.*

*Proof* When no pair of jobs exist for which the condition holds, whereas solving the LP-relaxation yields a double-nested solution, then we can convert it into a non-double-nested solution by combining Lemmas 4 and 5. We can then apply the Selection Algorithm to try to convert this nested, fractional solution to an integral solution with equal cost; if the Selection Algorithm fails, then we apply the perturbation technique by Verma and Dessouky (1998). □

The theorem encompasses two special cases, which are known to be solvable in polynomial time: Baptiste (2000) presents a polynomial time algorithm for the $1|r_j, p_j = p|\sum T_j$ problem, and it can easily be seen that the $1|p_j = p|\sum w_j T_j$ problem can be solved as an assignment problem.

### 5.1 Branching rule

First we define a pair of two jobs $J_{j_1}$ and $J_{j_2}$ to be *complicating* jobs if their weights, due dates, and release dates satisfy the following condition:

$$d_{j_1} < d_{j_2}, \qquad w_{j_1} < w_{j_2}, \quad \text{and} \quad r_{j_1} > r_{j_2}.$$

Suppose that we have found a fractional optimal solution. If it contains two complicating jobs $J_{j_1}$ and $J_{j_2}$ that are double nested, and in which at least one of the jobs has a tardy assignment, then we will branch. In all other cases, we first convert it to a non-double nested solution, which we then try to convert into an integral one with equal cost by applying the Selection Algorithm, and if necessary, through the perturbation technique by Verma and Dessouky (1998).

We branch by creating two sub-nodes from the current node by dividing the execution interval of job $J_{j_2}$ into two parts:

- A node where job $J_{j_2}$ has deadline $r_{j_1} + p - 1$.
- A node where job $J_{j_2}$ has release date $r_{j_1}$.

This means that job $J_{j_2}$ will either start before the release date of job $J_{j_1}$ or it will start at or after it. In the first case we ensure that the two jobs cannot be processed in the same time interval, whereas in the second case we create a new version of the problem where the release dates of the two jobs have become equal and thus in this new version these two jobs are not complicating jobs anymore. Note that a deadline for job $J_j$ can be incorporated by putting $x_{jt} = 0$ for all intervals that end after the deadline; this addition does not influence the results by Verma and Dessouky (1998).

Usually, there are several possibilities for selecting the pair of complicating jobs to branch on. We select to branch on the pair of complicating jobs that span the largest number of jobs, which is defined as the number of (partial) assignments to intervals that lie between the first completion interval and the last completion interval of the two complicating jobs. The larger this number, the more jobs are influenced by branching on this complicating pair of jobs.

## 6 Related objective functions

We have also looked at other objective functions besides total weighted tardiness, and we have found that the same approach based on the one by Verma and Dessouky (1998) can also be used for the objective functions:

– Total weighted number of late jobs
– Total weighted late work

We will discuss both of them in more detail now.

When we look at the weighted number of late jobs problem ($1|r_j, p_j = p|\sum w_j U_j$ in the three-field notation scheme), the cost of assigning job $J_j$ to time $t$ is:

$$c_{j,t} = \begin{cases} w_j & \text{if } t > d_j, \\ 0 & \text{otherwise.} \end{cases}$$

Our approach is again that we solve the LP-relaxation and, if fractional, try to convert the solution into a non-double-nested one with equal cost from which we then derive an optimal integral solution. Unfortunately, a solution in which $J_{j_1}$ and $J_{j_2}$ are nested in each other can be better than a non-double-nested solution in case $d_{j_1} < d_{j_2}$ and $r_{j_1} > r_{j_2}$, and we have to branch then. Baptiste (1999) presents a dynamic programming algorithm that solves this problem in polynomial time ($\mathcal{O}(n^7)$). Although according to the worst-case running time our algorithm might not perform as well, on average it could be very competitive since in case of dynamic programming the average and worst-case running times are equal.

We apply the same analysis as in Sect. 5 to remove double nestings, if possible. Suppose that the jobs $J_{j_1}$ and $J_{j_2}$ are double nested. The proofs for Lemmas 4 and 5 also hold for this objective function with the following two changes:

– In some of the cases there is not a strict decrease in cost by exchanging between two intervals, but the resulting schedule after the exchange will have equal cost. This is due to the binary character of this objective function.
– The case where $w_{j_1} \geq w_{j_2}$ cannot always be rewritten in the 2121 situation. The reason for this is again the binary character of the objective function. In case of total weighted tardiness an exchange between two tardy intervals never increases the objective function, since $d_{j_1}$ is strictly smaller than $d_{j_2}$. In case of weighted number of late jobs it is not guaranteed that this exchange does not increase the objective function. An example of a situation where the objective function increases is the same as the one depicted in Fig. 5 in Sect. 5. An exchange between the last two intervals will increase the cost because job $J_{j_1}$ will still be tardy and job $J_{j_2}$ will become tardy after the exchange. An exchange between the middle two intervals will also increase the cost since job $J_{j_1}$ will become tardy while job $J_{j_2}$ will still be on time after the exchange.

We see that regardless of the weights, the 2121 situation cannot always be rewritten. Therefore, we have to redefine the definition of complicating jobs for this objective function as follows. A pair of two jobs $J_{j_1}$ and $J_{j_2}$ are complicating jobs if their due dates and release dates satisfy the following condition:

$$d_{j_1} < d_{j_2} \quad \text{and} \quad r_{j_1} > r_{j_2}.$$

We need to extend the branching rule a little. From our experiments we saw that some instances took very long to solve. Looking at the branching information we saw that this was caused by jobs with a small time interval for being on time (i.e. a job $J_j$ with $d_j - r_j < 2p$). If such a job $J_j$ existed and was partially assigned to an on-time interval and a tardy interval, then job $J_j$ would be always selected for possible branching with another job $J_{j'}$, where $J_j$ and $J_{j'}$ are a pair of complicating jobs. To try and prevent such unnecessary branching, we changed the branching rule in the following way: In a node we first check whether there exists a job $J_j$ with $d_j - r_j < 2p$ and both tardy and non-tardy completion times. If such a job exists, we create two child nodes from the current node:

– A child node where job $J_j$ will be on time ($d_j$ is turned into a deadline);
– A child node where job $J_j$ will be tardy (the release date is increased to $d_j - p + 1$).

If such a job does not exist, we go on with the earlier suggested branching rule.

When we look at the weighted total late work problem $1|r_j, p_j = p|\sum w_j V_j$, the cost for assigning a job $J_j$ to time $t$ is:

$$c_{j,t} = \begin{cases} w_j p_j & \text{if } t \geq d_j + p, \\ w_j \max(0, t - d_j) & \text{otherwise.} \end{cases}$$

Since this objective in its most extreme form (when $C_j > d_j + p$ for a job $J_j$) behaves similarly to the weighted number of late jobs (cost will not increase anymore by placing job $J_j$ even later), we find that also for this objective function we cannot guarantee that the 2121 situation can always be rewritten into a nested solution of equal or smaller cost; the counterexample is the same as the one given for the sum of weighted late jobs, depicted in Fig. 5. This means that also for this objective function we have to redefine the definition of complicating jobs in the same way as for the weighted number of late jobs.

For both of these objective functions the case where the release dates and the due dates are equally ordered means that no complicating jobs can exist, which means that the solution to the LP-relaxation will either be already integral, or the fractional solution can be converted to an integral solution of same cost. This means that the cases where such an equal ordering exists are polynomially solvable.

## 7 Parallel identical machines

A nice advantage of the time-indexed formulation is that it can be used to solve problems with $m$ parallel, identical machines as well. In this machine environment, there are $m$ machines available, and processing job $J_j$ can be done by any one of these machines, which takes an uninterrupted period of length $p_j = p$ independent of the machine that executes it. We can formulate the problem as an ILP problem by using assignment variables $x_{j,t}$, which model our decision of executing job $J_j$ in period $t$. The only difference with the single-machine variant is that we now can process $m$ jobs at the same time, which we can model by adjusting the right-hand side in constraint (2) from 1 to $m$. This leads to the following ILP-formulation:

$$\min z = \sum_{j=1}^{n} \sum_{t \in A_j} c_{j,t} x_{j,t}$$

subject to

$$\sum_{t \in A_j} x_{j,t} = 1 \quad \text{for all } j \in N, \tag{4}$$

$$\sum_{j=1}^{n} \sum_{t' \in H_t \cap A_j} x_{j,t'} \leq m \quad \text{for all } t \in K \cup G, \tag{5}$$

$$x_{j,t} \in \{0,1\} \quad \text{for all } j \in N, t \in A_j. \tag{6}$$

It is readily verified that any integral solution can be converted into a feasible schedule by greedily assigning job $J_j$ to any machine that is available in the period $t$ for which $x_{j,t} = 1$.

Since the constraint matrix has not been changed, Lemma 1 still holds. Hence, we come to the following corollary.

**Corollary 1** *If any double-nested solution to the LP-relaxation is either sub-optimal or can be rewritten to a nested solution with equal cost, then there exists an optimal solution to the LP-relaxation that is integral.*

Since the proofs that we have given in the Sects. 4, 5, and 6 to show how to modify double-nested solutions do not depend on the number of machines involved, it follows that all our previous results concerning double-nested solutions still hold. Therefore, the only thing left to do is to find the integral optimal solution in case that solving the LP-relaxation yields a fractional one. Unfortunately, the heuristic presented in Sect. 3 for the single-machine case to convert a nested, fractional solution into an integral solution with equal cost could not be generalized to the $m$-machine case. Therefore, we immediately apply the perturbation method presented by Verma and Dessouky (1998).

## 8 Column generation

A big disadvantage of our time-indexed formulation is that it uses huge amounts of memory. For the biggest of the problems we tested memory usage was in the order of about 1.7 Gigabyte. To see if it was possible to reduce the amount of memory needed for solving the problems we looked at column generation.

The idea here is to split up the time horizon into $B$ time frames. These time frames do not necessarily need to have the same length but must have a length $\geq p-1$. This idea of dividing the complete time horizon has been independently suggested by Bigras et al. (2005) for solving the $1 \parallel w_j T_j$ problem by means of column generation. All time frames combined should cover the complete time horizon completely, and consecutive time frames may overlap at most $p-1$ time units. Each time frame $b$ ($b = 1, \ldots, B$) has a start time $\mu_b$ and end time $\omega_b$. Without loss of generality we created the time frames in such a way that two consecutive time frames have an overlap of exactly $p-1$ (i.e. $\mu_{b+1} + p - 1 = \omega_b$).

Given a division of the time horizon into a set of time frames, we compute for each time frame a feasible schedule for a subset of the jobs, such that each job is executed in exactly one time frame. We formulate this problem as an ILP using the concept of *time-frame plans*. A time-frame plan for a given time frame consists of the completion times for a subset of the jobs. The completion times must be feasible within the time frame, meaning that the jobs can actually be completed at their respective completion times, and all completion times must lie within the boundaries of the time frame. Now we need to find a suitable time-frame plan for each of the time frames. We need to make sure that all jobs are processed in exactly one of the selected time-frame plans. Furthermore, we also have to make sure that the total idle time in the overlapping part of two consecutive time-frame plans is greater than or equal to $p-1$, ensuring that we do not violate the capacity constraint in the overlapping part.

Let $s$ denote the number of possible time-frame plans. For each time-frame plan $i$ we determine the indicators $y_{ji}$ and $q_{bi}$, which are defined as

$$y_{ji} = \begin{cases} 1 & \text{if job } J_j \text{ is processed in time-frame plan } i, \\ 0 & \text{otherwise,} \end{cases}$$

$$q_{bi} = \begin{cases} 1 & \text{if time-frame plan } i \text{ is for time-frame } b, \\ 0 & \text{otherwise.} \end{cases}$$

We denote the cost of time-frame plan $i$ by $c_i$, and we use $y_i$ and $z_i$ to denote the amount of idle time in the front and end overlap of the time frame. We introduce the binary variable $x_i$ to indicate whether or not we include time-frame

plan $i$ in our solution. Now the complete ILP model is as follows:

$$\min \sum_{i=1}^{s} c_i x_i$$

subject to

$$\sum_{i=1}^{s} y_{ji} x_i = 1, \quad j = 1 \ldots n, \tag{7}$$

$$\sum_{i=1}^{s} q_{bi} x_i = 1, \quad b = 1 \ldots B, \tag{8}$$

$$\sum_{i=1}^{s} z_i q_{bi} x_i + \sum_{i=1}^{s} y_i q_{b+1,i} x_i \geq p - 1, \quad b = 1 \ldots B - 1, \tag{9}$$

$$x_i \in \{0, 1\} \quad \forall i. \tag{10}$$

Constraint (7) ensures that all jobs will be present in exactly one selected time-frame plan and constraint (8) ensures that for each time-frame exactly one time-frame plan will be selected. Finally constraint (9) ensures that the idle time in the overlap between two consecutive time-frame plans is at least $p - 1$.

To approximate the optimum of this integral model, we first relax the integrality constraint (10) to $x_i \geq 0$ (the other constraints ensure $x_i \leq 1$). We solve the resulting LP-relaxation through column generation.

In each iteration we only allow a subset of the time-frame plans and solve the LP-relaxation for this restricted set of variables. We then solve the corresponding *pricing* problem to find out whether there are time-frame plans the addition of which will reduce the objective function. The pricing problem is defined as finding the feasible time-frame plan with minimum reduced cost; if this minimum is non-negative, then we have solved the LP-relaxation to optimality. To compute the reduced cost we need the dual multipliers.

For each type of constraint in the master problem we get a dual multiplier:

- From (7) we get a dual multiplier $\pi_j$ for each job $J_j$.
- From (8) we get a dual multiplier $\tau_b$.
- From (9) we get a dual multiplier $\phi_b$ for each overlap between time frame $b$ and $b + 1$.

We solve the pricing problem for each time frame individually. It can be solved in a similar fashion as the original problem: for a given time frame we define the set of possible completion times and then assign these to eligible jobs such that the capacity constraints are satisfied. The changes that have to be made for solving the pricing problem for a given time frame $b$ are:

- We can discard any completion time $< \mu_b + p$ or $> \omega_b$ and create the following sets:
  - $A_j^b = \{a \in A_j | \mu_b + p \leq a \leq \omega_b\}$ $(j = 1, \ldots, n)$ denoting the possible completion times of job $J_j$ within time frame $b$.
  - $H_t^b = \{h \in H_t | \mu_b \leq h \leq \omega_b + p\}$ $\forall t$ denoting the time indices within time frame $b$ that are conflicting with time $t$.
  - $G^b = \{g \in G | \mu_b \leq g \leq \omega_b + p\}$ denoting all first possible completion times of all jobs within time frame $b$.
  - $K^b = \{k \in K | \mu_b \leq k \leq \omega_b + p\}$ denoting all other possible completion times of all jobs that are within time frame $b$.
- We are interested in jobs that are processed in the time frame $b$. Possibly not all jobs can be processed, so we relax the constraint 'all jobs must be processed exactly once' to 'all jobs must be processed at most once'.
- Since we need to know something about the available idle time in the overlapping regions, we add two extra jobs $\alpha_b$ and $\beta_b$, both with processing time $p$, for the front overlap and the end overlap of a time frame respectively. The possible completion times for these two extra jobs are:
  - $A_\alpha^b : \{\mu_b, \mu_b + 1, \ldots, \mu_b + p - 1\}$.
  - $A_\beta^b : \{\omega_b + 1, \omega_b + 2, \ldots, \omega_b + p\}$.

  It can be seen that these jobs have a special status, since job $\alpha$ starts before the start time of the time frame and job $\beta$ ends after the end time of the time frame. For the first time frame job $\alpha$ does not exist and for the last frame job $\beta$ does not exist because there is no time frame to overlap with.

Solving the pricing problem should result in a time-frame plan with minimum reduced cost. Hence, we need to set the cost of job $J_j$ being completed at time $t$ such that it corresponds exactly to its contribution to the reduced cost. If a job $J_j$ is not assigned in this time frame, it does not contribute to the cost function; otherwise, the cost $c_{j,t}$ of assigning job $J_j$ to time $t$ is defined as follows:

$$c_{j,t} = w_j \max\{0, t - d_j\} - \pi_j.$$

Furthermore the cost for assigning the $\alpha_b$ and $\beta_b$ jobs to a time $t$ are as follows:

$$c_{\alpha_b,t} = (t - \mu_b)\phi_{b+1},$$

$$c_{\beta_b,t} = (\omega_b + p - t)\phi_b,$$

which corresponds to the amount of idle time at the front or the back of the time frame multiplied by the corresponding dual multiplier.

The pricing problem we need to solve for a certain time frame $b$ is as follows:

$$\min \sum_{j=1}^{n} \sum_{t \in A_j^b} c_{j,t} x_{j,t} - \tau_b - \sum_{t \in A_\beta^b} c_{\beta_b,t} x_{\beta_b,t} - \sum_{t \in A_\alpha^b} c_{\alpha_b,t} x_{\alpha_b,t}$$

subject to

$$\sum_{t \in A_j^b} x_{j,t} \leq 1 \quad \text{for all } j \in N, \tag{11}$$

$$\sum_{t \in A_\alpha^b} x_{\alpha_b,t} = 1, \tag{12}$$

$$\sum_{t \in A_\beta^b} x_{\beta_b,t} = 1, \tag{13}$$

$$\sum_{j=1}^{n} \sum_{t' \in H_t^b \cap A_j^b} x_{j,t'} + \sum_{t' \in H_t^b \cap A_\alpha^b} x_{\alpha_b,t'} + \sum_{t' \in H_t^b \cap A_\beta^b} x_{\beta_b,t'} \leq 1$$

$$\text{for all } t \in K^b \cup G^b, \tag{14}$$

$$x_{j,t} \in \{0,1\}, \quad \text{for } j \in N \cup \{\alpha, \beta\}, t \in A_j^b. \tag{15}$$

Constraint (11) ensures that all regular jobs are processed at most once and constraint (12) and constraint (13) ensures that the alpha and beta job are processed exactly once. Finally constraint (14) ensures that there does not exist a time where more than one job is processed.

When the pricing problem is solved we know which jobs are assigned to which time intervals and we can calculate the cost $c_i$ of the new variable $x_i$ in the master problem as follows

$$\sum_{j=1}^{n} \sum_{t \in A_j^b} c_{j,t} x_{j,t}.$$

Furthermore the two indicators $y_i$ and $z_i$ denoting the idle time in the front overlap region and the rear overlap region can be set according to the assignment of the $\alpha$ and $\beta$ job.

For creating the initial variables for the master problem we ordered the jobs on their release dates and assigned them to the earliest possible time interval. From this integral feasible schedule we determined which jobs were processed in which time intervals and from this we created the initial variables. When the LP-relaxation is solved to optimality with the column generation, we add the integrality constraint back again to the master problem and then try to solve the master problem again with the generated set of columns.

Initial tests with a prototype implementation showed that the solutions given by solving the LP-relaxation with column generation often had a fractional solution value lower than the integral solution we found with our initial representation. As expected, the total amount of memory needed for solving the problems was a lot less because only a small part of the total number of variables was created in memory. One major disadvantage the tests showed was that the running time for solving the LP-relaxation to optimality grew considerably. However, since the problem is divided into a set of smaller, independent, subproblems, we can easily parallelize the solving of the subproblems. Since this LP-relaxation gave a lot weaker lower bound, one approach to solve the ILP problem to optimality would be to make use of branch-and-price. We did not implement this.

## 9 Computational experiments

For testing the time-indexed formulation, the branching rule, and the Selection Algorithm a program was written in Java. All tests were run on a Pentium 4 running at 3.00 GHz with 1 GB of RAM. For solving the ILPs we used Cplex 9.1 (ILOG 2005). We wanted to create some problems varying in size and did this by choosing the number of jobs $n$ from the set $\{70, 80, 90, 100\}$ and the common processing time $p$ from the set $\{5, 10, 15, 20, 25, 30\}$. For each combination of these two parameters we created 50 instances in the following way:

- $r_j$ was randomly selected from the interval $[0, (n-6)p)$.
- $w_j$ was randomly selected from the interval $[0, 120)$.
- $d_j$ was randomly selected from the interval $[r_j + p, (n-5)p)$.

This resulted in a total of 1200 instances. Each of these instances was then solved for the three objective functions sum of weighted tardiness, sum of weighted late jobs, and sum of weighted late work.

The results of the tests for the weighted tardiness problem can be found in Table 2. The first two columns denote the number of jobs and the common processing time. The next two columns give the average time in milliseconds needed for solving the instances by using pure Cplex without any extra information in the first column and by using Cplex enhanced with the branching rule and Selection Algorithm in the second. The following two give the average number of nodes that needed to be explored before the optimum was found. The next two give the average number of iterations needed before the problem was solved to optimality. The final column gives the average integrality gap for the set of instances.

One thing that can be clearly seen from the table is that the average integrality gap is very little to completely nonexistent, denoting that the relaxed time-indexed formulation gives a really strong bound for the solution. This can also be seen by looking at the average number of nodes that needed to be explored before the optimum was found.

**Table 2** Results weighted tardiness

| Jobs | $P$ | Average time for solving (ms) | | Average number of nodes | | Average number of iterations | | Average integrality gap (‰) |
|---|---|---|---|---|---|---|---|---|
| | | Cplex | B.R. | Cplex | B.R. | Cplex | B.R. | |
| 70 | 5 | 1891.60 | 1573.10 | 0.12 | 0.00 | 985.08 | 944.72 | 0.8 |
| 70 | 10 | 4780.08 | 4036.70 | 0.00 | 0.00 | 1721.20 | 1641.18 | = |
| 70 | 15 | 10220.30 | 6946.68 | 0.14 | 0.00 | 2021.24 | 1853.60 | 0.0 |
| 70 | 20 | 14435.70 | 10444.10 | 0.16 | 0.00 | 2364.98 | 2213.54 | = |
| 70 | 25 | 19684.76 | 15678.44 | 0.22 | 0.04 | 2721.16 | 2562.24 | 0.1 |
| 70 | 30 | 27206.66 | 17545.42 | 0.82 | 0.04 | 2897.80 | 2584.78 | 0.2 |
| 80 | 5 | 2840.56 | 2264.26 | 0.48 | 0.00 | 1334.74 | 1251.34 | 0.2 |
| 80 | 10 | 7877.82 | 5957.82 | 0.00 | 0.00 | 2002.34 | 1876.98 | 0.0 |
| 80 | 15 | 14807.52 | 10731.10 | 0.10 | 0.00 | 2454.92 | 2304.52 | 0.2 |
| 80 | 20 | 27111.76 | 16999.60 | 0.30 | 0.00 | 3043.64 | 2756.76 | 0.0 |
| 80 | 25 | 36674.16 | 20459.46 | 0.48 | 0.00 | 3236.98 | 2804.32 | = |
| 80 | 30 | 37105.58 | 25986.88 | 0.24 | 0.02 | 3363.52 | 3075.04 | 0.2 |
| 90 | 5 | 4144.52 | 2926.86 | 0.00 | 0.00 | 1527.34 | 1404.26 | 0.0 |
| 90 | 10 | 13632.42 | 9153.16 | 0.20 | 0.02 | 2458.74 | 2251.18 | 0.1 |
| 90 | 15 | 25802.94 | 15212.90 | 0.40 | 0.00 | 3146.56 | 2758.06 | = |
| 90 | 20 | 37260.04 | 23841.70 | 0.32 | 0.00 | 3578.20 | 3257.10 | = |
| 90 | 25 | 47910.14 | 31606.50 | 0.20 | 0.00 | 3939.44 | 3579.86 | = |
| 90 | 30 | 72489.82 | 42054.82 | 0.80 | 0.00 | 4322.44 | 3817.04 | = |
| 100 | 5 | 5153.96 | 4183.32 | 0.00 | 0.00 | 1754.06 | 1674.80 | 0.0 |
| 100 | 10 | 16928.82 | 12230.32 | 0.30 | 0.00 | 2805.62 | 2608.72 | = |
| 100 | 15 | 32375.18 | 23278.04 | 0.18 | 0.00 | 3566.20 | 3346.54 | 0.0 |
| 100 | 20 | 61929.32 | 34486.50 | 1.02 | 0.00 | 4463.46 | 3873.96 | = |
| 100 | 25 | 100738.12 | 50501.48 | 1.12 | 0.00 | 5326.60 | 4302.40 | 0.0 |
| 100 | 30 | 122807.82 | 75730.52 | 0.44 | 0.08 | 5655.16 | 4979.72 | 0.2 |

Cplex = Results Cplex 9.1, B.R. = Results with branching rule and Selection Algorithm = : Integrality gap was 0 for all instances

If we look at the first line (70 jobs with $P = 5$) in Table 2 we see that with the branching rule and Selection Algorithm on average 0 nodes need to be explored before the optimum is found, while there still is an integrality gap. The reason for this is that after solving the root node, Cplex added some cuts, after which the Selection Algorithm could convert the new solution to an integral solution. In the majority of the instances, after Cplex finished solving the root-node LP and possibly added some cuts, the Selection Algorithm could convert the found fractional solution to an integral solution of same cost. In the cases the Selection Algorithm could not be applied, the number of nodes that needed to be explored to find the optimum never exceeded 4. The number of nodes that were explored when only Cplex was used was at most 30 and for all instances this number was always greater than or equal to the number of nodes needed with the branching rule and Selection Algorithm.

The results for the objective functions weighted late jobs and weighted late work can be found in Tables 3 and 4 respectively.

If we look at these two tables then we see that for some of the problems, Cplex on average needs to explore fewer nodes before the optimum is found than our branch-and-bound algorithm. In all cases this increased average is caused by a single instance for which our branching rule branches on an unfortunate set of variables. This causes a lot of nodes to be explored before the optimum can be found.

Another observation is that for the number of weighted late jobs problem, for two of the sets of larger instances there exist instances that Cplex could not solve. Initially these instances were not solvable with the original branching rule either; only after changing the branching rule as mentioned in Sect. 6 these instances could be solved.

The implementation was not optimized for speed and some improvements are still possible. These changes would

**Table 3** Results weighted late jobs

| Jobs | $P$ | Average time for solving (ms) | | Average number of nodes | | Average number of iterations | | Average integrality gap (‰) |
|---|---|---|---|---|---|---|---|---|
| | | Cplex | B.R. | Cplex | B.R. | Cplex | B.R. | |
| 70 | 5 | 2173.76 | 1614.40 | 0.28 | 0.00 | 925.26 | 824.52 | 19.0 |
| 70 | 10 | 6336.38 | 4704.88 | 0.22 | 0.04 | 1584.54 | 1410.76 | 5.2 |
| 70 | 15 | 15812.24 | 9536.66 | 24.12 | 0.18 | 1852.96 | 1590.34 | 11.9 |
| 70 | 20 | 19019.84 | 14309.14 | 2.08 | 4.06 | 2177.22 | 2005.26 | 7.1 |
| 70 | 25 | 32457.72 | 20199.04 | 33.20 | 0.14 | 2592.64 | 2149.72 | 18.1 |
| 70 | 30 | 32067.36 | 21591.08 | 0.50 | 0.10 | 2335.52 | 2082.78 | 4.0 |
| 80 | 5 | 4115.68 | 2787.94 | 17.38 | 0.14 | 1251.30 | 1098.20 | 14.1 |
| 80 | 10 | 11372.56 | 6780.58 | 14.86 | 0.04 | 1903.24 | 1623.14 | 1.3 |
| 80 | 15 | 19108.02 | 11399.24 | 0.36 | 0.00 | 2292.64 | 1943.30 | 0.8 |
| 80 | 20 | 39983.62 | 21466.40 | 23.50 | 0.34 | 2880.24 | 2369.90 | 23.1 |
| 80 | 25 | 57937.80 | 38194.18 | 2.58 | 0.36 | 3281.24 | 2534.72 | 13.1 |
| 80 | 30 | 56392.76 | 45024.90 | 2.48 | 15.24 | 3182.84 | 3000.28 | 4.7 |
| 90 | 5 | 5204.18 | 3042.52 | 0.56 | 0.00 | 1436.70 | 1226.82 | 11.6 |
| 90 | 10 | 18414.40 | 9015.18 | 1.04 | 0.00 | 2364.08 | 1901.80 | 3.5 |
| 90 | 15 | 45076.90 | 21474.64 | 19.16 | 0.12 | 3167.76 | 2452.88 | 17.0 |
| 90 | 20 | 55830.74 | 29861.64 | 0.94 | 0.18 | 3482.40 | 2812.38 | 12.9 |
| 90 | 25 | 96773.20 | 44125.78 | 40.16 | 0.14 | 4263.70 | 3109.56 | 0.9 |
| *90 | 30 | 210834.68 | 74404.14 | 365.88 | 0.60 | 6117.14 | 3750.76 | 23.2 |
| 100 | 5 | 6671.16 | 4945.88 | 1.38 | 0.10 | 1683.70 | 1495.80 | 9.0 |
| 100 | 10 | 36198.62 | 18203.18 | 48.52 | 0.20 | 3201.30 | 2416.08 | 10.9 |
| 100 | 15 | 53255.30 | 26856.22 | 1.64 | 0.04 | 3546.68 | 2911.34 | 7.3 |
| 100 | 20 | 125703.80 | 44362.66 | 3.72 | 0.24 | 4959.40 | 3313.70 | 13.9 |
| 100 | 25 | 189431.78 | 78217.88 | 356.50 | 0.84 | 7652.92 | 4092.86 | 4.2 |
| *100 | 30 | 374531.18 | 173092.26 | 326.68 | 1.04 | 11306.32 | 5049.76 | 21.4 |

Cplex = Results Cplex 9.1, B.R. = Results with branching rule and Selection Algorithm, *Cplex could not solve one or more instances

not improve the number of nodes or number of iterations for any of the problems, but would only decrease the time needed for solving the problems. The more nodes that need to be explored, the bigger the decrease in time would be.

Another observation that was made during the tests, is that the memory needed by pure Cplex was on average considerably more compared to the memory consumption of Cplex combined with our branching rule. This can be explained by the fact that Cplex branches on single variables, while our branching rule branches on a set of variables and thus the branching depth stays lower.

## 10 Conclusion and further research

Our basis problem is the $1|r_j, p_j = p|\sum w_j T_j$ problem. We have presented a time-indexed ILP-formulation for this problem and show that if the instance does not contain two jobs $J_i$ and $J_j$ such that $d_i < d_j$ and $w_i < w_j$ and $w_i < w_j$ then the LP-relaxation can be converted into an optimal solution. For the general case we present an efficient branch-and-bound algorithm.

Next, we showed that the same approach could be used for the single-machine problems of minimizing total weighted late work and total weighted late jobs, respectively. Here the LP-relaxation can be converted into an optimal solution in case the release dates and due dates are equally ordered.

We further showed that the results obtained for the single-machine problems can directly be translated to the corresponding problems with $m$ identical, parallel machines, except for that we could not generalize the single-machine Selection Algorithm to the $m$-machine situation. This can be circumvented by using small perturbations in the cost function.

**Table 4** Results weighted late work

| Jobs | $P$ | Average time for solving (ms) | | Average number of nodes | | Average number of iterations | | Average integrality gap (‰) |
|---|---|---|---|---|---|---|---|---|
| | | Cplex | B.R. | Cplex | B.R. | Cplex | B.R. | |
| 70 | 5 | 2080.66 | 1489.94 | 0.00 | 0.00 | 901.72 | 816.60 | 0.9 |
| 70 | 10 | 5456.00 | 3844.50 | 0.00 | 0.00 | 1534.30 | 1380.56 | 1.3 |
| 70 | 15 | 12834.56 | 7676.14 | 10.92 | 0.04 | 1868.04 | 1636.96 | 1.0 |
| 70 | 20 | 16126.78 | 10842.42 | 0.12 | 0.40 | 2077.12 | 1876.96 | 0.4 |
| 70 | 25 | 30571.46 | 19438.82 | 21.00 | 0.22 | 2511.44 | 2217.86 | 3.1 |
| 70 | 30 | 28914.50 | 18249.92 | 2.86 | 0.24 | 2495.60 | 2169.92 | 3.7 |
| 80 | 5 | 3395.70 | 2322.50 | 0.54 | 0.08 | 1237.94 | 1081.62 | 0.9 |
| 80 | 10 | 10167.48 | 6547.82 | 0.36 | 0.02 | 1971.60 | 1695.80 | 1.2 |
| 80 | 15 | 16948.26 | 11093.56 | 0.24 | 0.06 | 2254.70 | 2026.52 | 0.2 |
| 80 | 20 | 27690.58 | 17443.74 | 0.28 | 0.08 | 2679.92 | 2351.72 | 0.8 |
| 80 | 25 | 73342.68 | 31503.12 | 60.88 | 0.24 | 3513.04 | 2628.76 | 3.3 |
| 80 | 30 | 59537.10 | 26996.18 | 45.68 | 0.22 | 3220.96 | 2599.38 | 0.7 |
| 90 | 5 | 5406.16 | 2836.42 | 1.26 | 0.00 | 1463.48 | 1228.06 | 0.2 |
| 90 | 10 | 17117.94 | 10031.40 | 1.06 | 0.34 | 2462.54 | 2007.40 | 1.1 |
| 90 | 15 | 31236.90 | 15361.46 | 0.72 | 0.00 | 2821.76 | 2371.84 | 0.2 |
| 90 | 20 | 55840.70 | 24638.02 | 0.80 | 0.00 | 3455.44 | 2759.28 | 0.0 |
| 90 | 25 | 74709.72 | 34966.28 | 12.50 | 0.28 | 3983.92 | 3074.80 | 1.2 |
| 90 | 30 | 100073.20 | 48197.24 | 1.28 | 0.06 | 4391.50 | 3459.70 | 1.0 |
| 100 | 5 | 5781.44 | 4375.96 | 0.68 | 0.06 | 1617.74 | 1468.30 | 4.4 |
| 100 | 10 | 29789.30 | 13383.64 | 5.40 | 0.06 | 3111.58 | 2428.16 | 4.2 |
| 100 | 15 | 46590.86 | 32824.52 | 1.30 | 1.50 | 3574.30 | 3086.56 | 1.4 |
| 100 | 20 | 95535.26 | 36192.38 | 1.96 | 0.08 | 4732.06 | 3283.82 | 2.3 |
| 100 | 25 | 184365.38 | 67680.54 | 3.98 | 0.42 | 6221.94 | 4079.94 | 0.9 |
| 100 | 30 | 181203.72 | 84013.28 | 3.24 | 0.48 | 5599.68 | 4349.68 | 2.8 |

Cplex = Results Cplex 9.1, B.R. = Results with branching rule and Selection Algorithm

Finally, we have shown that the technique of column generation can be used to solve this kind of problems in case of a lack of memory to perform the standard ILP algorithm.

There are three clear open questions. The first one is to establish the computational complexity of $1|r_j, p_j = p|\sum w_j T_j$? Although we have given a good characterization of the problem, it is still an open question whether the general $1|r_j, p_j = p|\sum w_j T_j$ problem is polynomially solvable. The second question is to devise a fast, exact algorithm to convert a fractional, nested solution to an integral solution. The third question is to determine a heuristic or exact algorithm to convert a fractional, nested solution to an integral solution for the $m$-machine case.

**Acknowledgements** The authors want to express their gratitude to Chams Lahlou. As a referee, he gave very useful comments on an earlier draft of the paper in which he showed by example (which example we copied in Fig. 2) that the Selection Algorithm is just a heuristic. The research of the second author was conducted when he was a Ph.D.-student at Utrecht University.



## References

Akturk, M. S., & Ozdemir, D. (2001). A new dominance rule to minimize total weighted tardiness with unequal release dates. *European Journal of Operational Research*, *135*, 394–412.

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

Baptiste, P. (2000). Scheduling equal-length jobs on identical parallel machines. *Discrete Applied Mathematics*, *103*(1–3), 21–32.

Baptiste, P., Brucker, P., Knust, S., & Timkovsky, V. (2004). Ten notes on equal-processing-time scheduling. *4OR: Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, *2*, 111–127.

Bigras, L.-P., Gamache, M., & Savardan, G. (2005). Time-indexed formulations and the total weighted tardiness problem. Technical report, GERAD and 'Ecole Polytechnique de Montréal.

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

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

ILOG (2005). ILOG CPLEX v9.1, http://www.ilog.fr.

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

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

Leung, J. Y.-T. (2004). *Handbook of scheduling*. New York: Chapmann & Hall/CRC.

Nemhauser, G. L., & Wolsey, L. A. (1988). *Integer and combinatorial optimization*. *Wiley-interscience series in discrete mathematics and optimization*. New York: Wiley.

Roos, C. (2005). Private communication.

Verma, S., & Dessouky, M. (1998). Single-machine scheduling of unit-time jobs with earliness and tardiness penalties. *Mathematics of Operations Research*, *23*(4), 930–943.