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

G. Diepen

J.M. van den Akker

J.A. Hoogeveen

institute of information and computing sciences, utrecht university

technical report UU-CS-2005-054

www.cs.uu.nl

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

G. Diepen, J.M. van den Akker, and J.A. Hoogeveen

Department for Information and Computing Sciences,
Utrecht University, P.O. Box 80089, 3508 TB Utrecht, The Netherlands.
e-mail: {diepen,marjan,slam}@cs.uu.nl

Abstract. In this paper we look at the problem where we have to schedule n jobs with release dates, due dates, weights, and equal processing times on a single machine. The objective is to minimize the sum of weighted tardiness ($1|r_j, p_j = p|\sum w_j T_j$ in the three-field notation scheme). 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 rewritten 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 m identical, parallel machines variants.

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 we have a release date r_j before which job 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, we define the tardiness T_j of job 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, Lawler, Lenstra, and Rinnooy Kan (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 $1|r_j, p_j = p|\sum w_j T_j$.

Over the years quite some research has been done on scheduling problems regarding (weighted) tardiness. Lawler (1977) gave a pseudopolymial algorithm for solving the $1||T_j$ problem and Du and Leung (1990) gave a proof for the $1||T_j$ to be $\mathcal{NP}$-hard in the ordinary sense. The weighted version of this problem, $1||w_j T_j$, is known to be $\mathcal{NP}$-hard in the strong sense (Lenstra, Rinnooy Kan, and Brucker 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.

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

Also quite some research has been done on scheduling problems regarding 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, Brucker, Knust, and Timkovsky (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 then. In this paper we will use their approach, and we will show that for some special cases their analysis goes through, which implies that for these cases the problem is solvable in polynomial time.

The outline for the rest of this paper is as follows: In Section 2 we give an ILP-formulation of the problem. In Section 3 we will discuss the special case of the problem in which the jobs have a common due date. In Section 4 we will discuss the general problem, and in Section 5 we will apply the same techniques on problems with related objective functions. After that in Section 6 we will discuss another way of solving the problem, which aims at reducing the amount of memory needed, and in Section 7 we will present some experimental results. Finally in Section 8 we will draw some conclusions.

2 Problem formulation

Like in Verma and Dessouky (1998), we make use of a time indexed formulation for representing the problem as an ILP by taking into account only 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 exist an optimal schedule without unnecessary idle time.

Since there exists an optimal schedule without unnecessary idle time, each job in that schedule will either start at its release date, or right after another job. So each job introduces a set of possible completions times, and the extreme would be to start some job j right at its release date and place all other jobs immediately after it. This means that the number of possible completion times job j introduces is equal to $(n-1)$.

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

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

Furthermore for each job j we define $\gamma_j = r_j + p$ to be the first possible completion time of job j. Unless another job j' has a release date that is a multiple of p before r_j, only job j can be completed at time γ_j.

We define the set G denoting all first possible completion times of all jobs by

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

and the set K denoting all other possible completion times as

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

Since the first possible completion time of job j is γ_j, we find that the set of possible completion times of a job j is

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

Since the capacity of the machine is equal to 1, it means that when a job j is completed at time t, no other job 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

$$H_t := \{t' \in K \cup G | 0 \leq t - t' < p\}.$$

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

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

Furthermore we define the cost $c_{t,j}$ of having job 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

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

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

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

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 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 α_j is the earliness penalty for job j and β_j is the tardiness penalty for job j. Since they consider earliness penalties, sometimes it can be beneficial to introduce idle time before starting to process a job. They show that when the jobs can be indexed such that $\alpha_1 \leq \alpha_2 \leq \ldots \leq \alpha_n$ and $\beta_1 \leq \beta_2 \leq \ldots \leq \beta_n$, the LP-relaxation of the time indexed formulation always yields an integral solution. Before we go further into theory we need some definitions.

Definition 1. *Let x be a feasible solution to the LP-relaxation. We say that job j_2 is* nested *in job j_1 ($j_1 \neq j_2$), if there exists some $t_k \in K \cup G$, ($k = 1, 2, 3$) 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.*

We now define the $\prec$ relation based on the above definition as follows: $j_1 \prec j_2$, if a job j_1 is nested in job j_2.

Definition 2. (Verma and Dessouky 1998) *A feasible solution of the LP-relaxation is non-nested if and only if there exists a pair of jobs j_1 and j_2 which are nested in each other, i.e., $j_1 \prec j_2$ and $j_2 \prec j_1$. If no such pair of two jobs exists, the solution is a nested solution.*

For ease of writing we furthermore define the notation 1212 meaning 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.

Verma and Dessouky (1998) give a proof for the $1|p_j = p| \sum \alpha_j E_j + \beta_j T_j$ to be polynomially solvable. This proof was divided into two parts: First they prove that for their objective function a non-nested schedule would always be sub-optimal. Second, they prove that if an optimal

schedule cannot be non-nested (and thus must be nested), all extremal optimal solutions to the LP-relaxation are integral in case there exist no two jobs with equal weights, and that there exists an extremal integral optimal solution otherwise. This second proof is independent of the objective function and is solely based on the fact that if each optimal solution is nested, 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).

Using the second result given by Verma and Dessouky (1998), we only have to show that non-nestedness will always yield a sub-optimal solution.

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 Figure 1. The maximum capacity of the machine is equal to 1 (i.e. at most 1 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 job j_2 and j_3 each have fractional allocation values, and hence they are both partially allocated to two different time intervals.

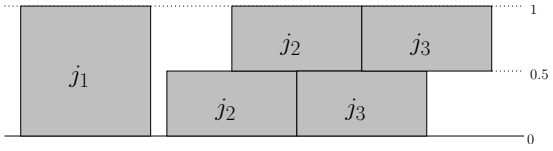

Fig. 1. Example of a fractional solution in the LP-relaxation

Lemma 1. *The results of Verma and Dessouky (1998) are not influenced by adding release dates.*

Proof. Verma and Dessouky (1998) show that if a non-nested solution is sub-optimal, the constraint matrix for the non-zero columns is totally unimodular. Release dates can be modelled by adding a constraint for each pair of a job and a time index, with the time index being before the first possible completion time of the job, that fixes the corresponding variable to zero. 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. □

Now we can observe the following:

Corollary 1. *The following analysis also holds for the case where we have m parallel identical machines.*

Proof. The problem with m parallel identical machines can be obtained by replacing the ≤ 1 in Constraint (2) by $\leq m$. This substitution does not influence the constraint matrix, only the right hand side is changed. Furthermore the definition of non-nested stays exactly the same. □

3 Common due date

In this section we take a look at the situation where all jobs j have a common due date $d_j = D$. First in Section 3.1 we take a 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 can adjust the common due date back to 0, resulting in the $1|r_j, p_j = p, d_j = 0|\sum w_j T_j$ problem, which is equivalent to the $1|r_j, p_j = p|\sum w_j C_j$ problem. Baptiste (2000) already showed that this problem could be solved in polynomial time ($\mathcal{O}(n^7)$) by means of dynamic programming.

In Section 3.2 we take a look at the case where $D \geq r_{\min} + p$. In this case it is possible to complete at least one job before the due date.

For both cases we assume without loss of generality that the jobs are reindexed such that $w_{j_1} \leq w_{j_2} \leq w_{j_3} \leq \ldots \leq w_{j_n}$. If after reindexing the jobs $w_{j_1} < w_{j_2} < w_{j_3} < \ldots < w_{j_n}$ holds, we say that a strict order on the weight of the jobs exists.

3.1 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 non-nested. Therefore, according to Definition 2 there exist jobs j_1 and j_2 which 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_1 is nested in job 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 $\varepsilon = \min(x_{j_2,t_1}, x_{j_1,t_2}, x_{j_2,t_3})$.

In Figure 2 an example of the situation is depicted where without loss of generality we assumed $\varepsilon = x_{j_2,t_3}$. Job j_1 is nested in job j_2, and job j_2 is nested in job j_1. Since both jobs are tardy, the exchange of ε between x_{j_1,t_2} and x_{j_2,t_3} causes a change in the cost of Δ, where we have:

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

In the resulting situation job j_2 is still nested in job j_1, but job j_1 is not nested in job j_2 anymore. The exchange does not violate any constraint as the amount of total allocation at each interval is unaltered, 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 $\Delta < 0$.

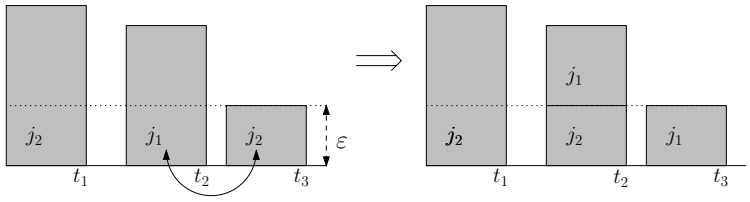

Fig. 2. Example of rearranging ε between two nested jobs.

If there is no strict order on the weight of the jobs, then the optimal schedule might be non-nested. However, this non-nested schedule can always be rewritten into a nested schedule with same cost, which can be rewritten into a feasible schedule with same cost. This can be easily seen because when two jobs have equal weights, their parts can be exchanged arbitrarily while the total cost stays the same, yielding a nested solution with equal cost. □

Hence, we see that the solution to the LP may be fractional due to the mingling of parts of jobs with equal weights. Such a fractional solution can be rewritten to an integral solution as

described above. Verma and Dessouky (1998) use another approach and show that by adding a small perturbation to the coefficients in the objective function the solution to the LP will always be integral. This is done in the following way:

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

where ε is a very small positive number.

We have now shown that for the problem $1|r_j, p_j = p| \sum w_j C_j$ a non-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. 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 $\mathcal{O}(n^{13}\sqrt{n}\log\frac{n^3}{\epsilon})$. Although worst-case this is a more than the previous known result of Baptiste (2000) with dynamic programming, on average it might be a lot quicker.

3.2 Common due date ($d_j = D$)

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 non-nested optimal solution is either sub-optimal, or it can be rewritten into a nested solution with the same cost.*

Proof. Assume that we have an optimal non-nested solution. Since $j_1 \prec j_2$, there must exist intervals t_1, t_2, and t_3 all in set $K \cup G$ 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_1 and 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 same 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 non-nested solution into a nested solution. Otherwise we use the same approach as in the proof of Lemma 2, and we define $\varepsilon = \min(x_{j_2,t_1}, x_{j_1,t_2}, x_{j_2,t_3})$. The optimal way of allocating 2ε of job j_2 and ε of job j_1 among the three intervals is to first allocate ε of job j_2 to interval t_1, then allocate ε of job j_2 to interval t_2 and finally allocate ε of job j_1 to interval t_3. The idea behind this can be seen again in Figure 2. If there exists a strict order on the weights of the jobs, the resulting solution after the exchange will have smaller cost. If no strict order exists, then the resulting solution will be of equal cost if $w_{j_1} = w_{j_2}$. This implies that we can improve or rewrite the non-nested solution without violating any constraints as the amount of total allocation at each interval in unaltered, 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.

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

Again, due to the possible mingling of the non-tardy jobs it is possible to end up with a fractional solution, while an integral solution of same cost also exists. One way to solve this problem is again to add small perturbations to the cost coefficients in the objective function. Another possibility is to rewrite a fractional extremal solution in such a way that we end up with an integral extremal solution of same cost. We explain this rewrite strategy below. For this rewrite strategy we first need the following preliminary lemma.

Lemma 4. *The number of jobs that can be processed completely in the time span $[0, t]$, for any value of t, does not increase by relaxing the integrality constraints.*

Proof. The proof is done by contradiction. Assume we have a fractional solution in which the number of completely processed jobs is bigger than in the integral solution. Without loss of generality we assume that job j is the first job that is processed completely before t, but with fractional assignments; the first interval it is assigned to we denote with t'. Because the job is completely processed in the time span $[0, t]$, the rest of this job must be assigned to other intervals in the range $[t', t]$. Now we have to consider two situations:

- No other job is assigned to an interval that is less than p away from time t'. This means that there is still room to assign the rest of job j to interval t'.
- There exists another job that has a partial assignment to an interval t'', where $t'' - t' < p$. We can exchange the partial assignment of this second job with the other parts of the first job without changing the number of totally completed jobs. The resulting situation can be converted into one where job j is fully assigned to interval t'.

Now the same problem remains for the time span $[t', t]$. Continuing this process will result in an integral solution for the same jobs. Hence, the maximum number of completely processed jobs does not increase when the integrality constraints are relaxed and is at most equal to $\lfloor \frac{t}{p} \rfloor$ (if permitted by the release dates of the jobs), thus contradicting our assumption. □

Lemma 5. *If there is a strict order on the weights of the jobs, then in any optimal solution there does not exist a job j_1 that is partly tardy, that is, it has been partly assigned to intervals $t_1 \leq D$ and $t_2 > D$. If there is no strict order on the weights of the jobs then such a job j_1 can only exist if there exists another such job j_2 with $w_{j_1} = w_{j_2}$. In case two such jobs exist, we can rewrite the solution such that no job is partly tardy.*

Proof. The proof is done by contradiction. Assume we have an optimal solution with a job j_1 and $t_1 \leq D$ and $t_2 > D$ such that x_{j_1,t_1} and x_{j_1,t_2} are both positive.

Without loss of generality we reorder all jobs that are processed before the common due date D. Because all of these jobs are on time, it does not matter in what order they are, so we can reorder them based on their release dates. In case there was a job that was allocated to multiple completion times $\leq D$, this reordering now allocates this job to exactly one time. After the reordering the part of job j_1 that was allocated to t_1 is now allocated to some time $t_1' \leq D$.

Now we can make use of Lemma 4, which states that by relaxing the problem the number of jobs we can fully process stays the same. So after the reordering there are only two cases left to consider:

- All other jobs that are allocated to completion times $\leq D$ are all completely processed, and hence, according to Lemma 4, there must be still capacity left at time t_1'. Thus we can transfer the tardy allocation at x_{j_1,t_2} to $x_{j_1,t_1'}$, which improves the current solution. This contradicts our assumption that we had an optimal schedule.
- There exists another job j_2 and times $t_3 \leq D$ and $t_4 > D$ such that x_{j_2,t_3} and x_{j_2,t_4} are both > 0. In case $w_{j_1} \neq w_{j_2}$ (always true when a strict order on the weights of the jobs exists) we can always improve the current solution by exchanging the fractional parts of the two jobs in such a way that the job with the higher weight is moved to an earlier interval, contradicting our assumption we had an optimal schedule. In case $w_{j_1} = w_{j_2}$ (only possible when no strict order exists), we can rewrite the current solution by exchanging the fractional parts of the two equal-weight jobs such that one of the jobs is completed before D and the other one after D, resulting in a solution of same cost. If there exist more jobs with equal weights, we can repeat the above steps for pairs of jobs.

□

In case we find a fractional extremal solution we can rewrite it as follows:

- Reorder all jobs that are completed before the common due date based on their release dates.

– Reorder all equal-weight jobs that are completed after the common due date arbitrarily.

The proof for this rewrite rule follows directly from the proof of Lemma 5.

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

Theorem 1. *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 Lemma 2 and Lemma 3 we know that for any value of D there exists an optimal schedule that is nested. By making use of Lemma 1 we know that by solving the LP relaxation of the time-indexed formulation we either get an integral extremal solution or a fractional one. In case of a fractional extremal solution we know that there exists an integral solution of same cost. By adding a perturbation to the coefficients in the objective function or by using a rewrite rule based on the proofs of Lemma 3 and Lemma 5, we can find this integral solution with equal cost. Both adding the perturbation and using the rewrite rule yield an integral solution in polynomial time. □

4 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 and a rule for rewriting certain fractional solutions into integer solutions of the same cost.

Unlike the common due date case, with arbitrary due dates the LP solution can have a better value than the optimal integer solution. An example that shows this is the following instance:

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

Common processing time $p = 2$.

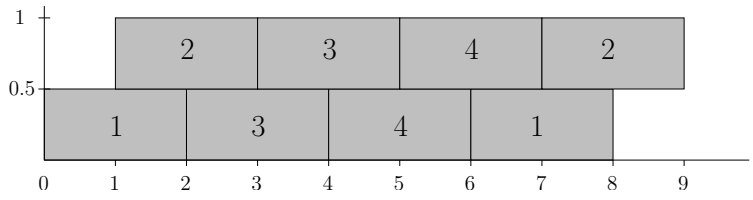

Fig. 3. Optimal LP solution that is non-nested

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

– start with job 1. After that process job 3 and job 4 and finally process job 2, which will be tardy. This solution has cost 5.
– start with job 2. After that one of the jobs 1,3,4 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 3 or 4. Since there is enough of capacity before job 3, job 1 can be placed in front of job 3 without delaying it, leading to the first situation and thus starting with job 3 can not be optimal.

Thus the optimal ILP solution value equals 5, which is different from the optimal LP solution with value 3. Furthermore we can see in Figure 3 that the jobs 1 and 2 are non-nested.

Now we want to look at the situation in which two jobs j_1 and j_2 are non-nested and we want to show in which cases such a non-nested situation is sub-optimal. Without loss of generality we assume $d_{j_1} < d_{j_2}$. In what follows we will show that the case of $d_{j_1} = d_{j_2}$ always yields an integer solution or a fractional solution that can be rewritten into an integer solution of same cost. This assumption leaves us with two possible non-nested situations for the two jobs:

- Situation 1212
- Situation 2121

Lemma 6. *In the 1212 situation where there are two jobs j_1 and j_2 with $d_{j_1} \leq d_{j_2}$, a non-nested solution is either sub-optimal, or it can be rewritten into a nested solution of the same cost.*

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

- Job j_2 will still be on time.
- Job 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 ϵ between x_{j_2,t_2} and x_{j_1,t_3}.
- $w_{j_1} < w_{j_2}$. Exchanging ε between x_{j_1,t_3} and x_{j_2,t_4} causes a change in cost of Δ, where we have:

$$\begin{aligned}\Delta &= w_{j_1}(t_3 - d_{j_1}) + w_{j_2}(t_4 - d_{j_2}) - w_{j_1}(t_4 - d_{j_1}) - w_{j_2}(t_3 - d_{j_2}) \\ &= w_{j_1}(t_3 - t_4) + w_{j_2}(t_4 - t_3) \\ &= (w_{j_2} - w_{j_1})(t_4 - t_3) \\ &> 0.\end{aligned}$$

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 7. *In the 2121 situation where there are two jobs j_1 and j_2 with $d_{j_1} < d_{j_2}$, a non-nested solution is 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 $d_{j_1} < d_{j_2}$. 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:

– $d_{j_2} \geq t_2$. In this case job 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_2 is still on time at time t_2 and job 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 $r_{j_1} \leq r_{j_2}$.
If $d_{j_1} < d_{j_2}$, $w_{j_1} < w_{j_2}$ and $r_{j_1} > r_{j_2}$ we cannot guarantee that we can always improve the non-nested situation 2121. An example of such a situation is depicted in Figure 4. If w_{j_2} is greater than or equal to $\frac{(t_4-t_3)w_{j_1}}{(t_4-d_{j_2})}$, exchanging between the two intervals will not be profitable.
– $d_{j_2} < t_2$. In this case both jobs are tardy at times t_2, t_3, and t_4. It will always be profitable to exchange x_{1,t_2} and x_{2,t_3}, because you want the job with the larger weight (j_2 in this case) more to the front.

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 $r_{j_1} > r_{j_2}$. □

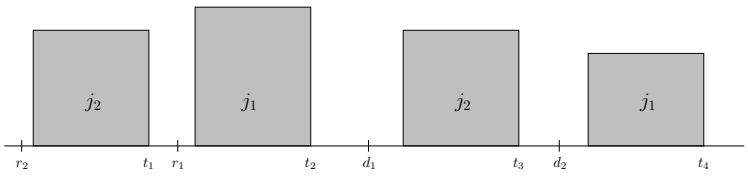

Fig. 4. Example of situation where rewriting is not profitable

Theorem 2. *When there are no two jobs j_1 and 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. If this condition does not hold the extremal solution to the LP can be fractional and branching may be needed.*

Proof. The proof for the first part follows directly out of combining Lemma 6 and Lemma 7. In the case we do end up with an fractional extremal solution to the LP while there does not exist a pair of jobs for which the condition holds, the problem can still be solved in polynomial time by adding a small perturbation to the coefficients in the objective function, or in some cases by using the rewrite rule presented in Section 4.2. Both ways yield a solution in polynomial time.

Otherwise it is clear that branching is needed for solving the problem. □

4.1 Branching rule

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

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

There is only need for branching when we have a fractional solution in which there are two complicating jobs j_1 and j_2 that are non-nested, and in which at least one of the jobs has a tardy assignment.

In all other cases there is no need for branching: we have a fractional extremal solution and by adding a small perturbation to the coefficients in the objective function we end up in an integral solution of equal cost. In some cases it will even be possible to make use of the rewrite rule that will be presented in Section 4.2 to find the integral solution without the need of adding perturbations.

So not only the case of a common due date (discussed in Section 3), but also the cases of a common release date and a common weight for all jobs are solvable in polynomial time, since there cannot exist a pair of complicating jobs. For both of these problems polynomial time algorithms

already exist; Baptiste (2000) gives 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.

When we have two jobs j_1 and j_2 that are complicating jobs, non-nested, and at least one of the parts of the jobs is tardy, then we need to branch. The way we branch is by creating two sub-nodes from the current node by dividing the execution interval of job j_2 into two parts:

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

This means that job j_2 will either start before the release date of job 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.

Given a node where branching is needed it is possible that multiple branching points are available. If there are multiple branching points available, 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.

4.2 Rewrite rule

As stated before, branching is only needed in case of a fractional solution with two complicating jobs j_1 and j_2 being non-nested, and with at least on of the jobs having a tardy assignment. In other cases, the fractionality is due to the fact that in the proofs of Lemma 6 and Lemma 7 there exist situations where there is no improvement when rewriting a non-nested solution to a nested one. The problem of ending up in such a fractional extremal solution can be solved by adding a small perturbation to the coefficients in the objective function.

In some cases, like with the common due date, it is possible to just rewrite a given fractional solution into an integer solution without the need of using perturbations. Below we present an algorithm to rewrite a fractional solution in which all fractional jobs are non-tardy into an integer solution with equal cost.

Rewrite Rule:

1. Fix all variables that have value 1 in the fractional solution at 1.
2. Determine the list of all possible completion times and filter out those times that have become impossible due to the fixing in Step 1.
3. Sort the jobs on their release dates.
4. While there are unfixed jobs left:
   (a) Determine job j with earliest release date.
   (b) Remove all possible completion times smaller than $r_j + p$.
   (c) Determine the earliest possible completion time C'.
   (d) Determine the set of jobs that have release date smaller than or equal to $C' - p$ (i.e. all jobs that can be completed at time C') and choose from this set the job with the earliest due date; we denote this job by q.
   (e) If there exists another job j^* for which:
   $C' - p < r_{j^*} < C'$
   and
   $d_{j^*} < C' + p$
   then assign job j^* to its first possible completion interval. If such a job j^* does not exist, then assign job q to interval C'.

Lemma 8. *The algorithm above will rewrite every fractional solution where all of the fractional parts of the jobs are non-tardy to an integer solution with equal cost.*

Proof. Assume that there is a fractional solution with cost K (where K comes from the already fixed jobs, because we have assumed all fractionally allocated jobs to be non-tardy) for which the rewrite rule finds an integer solution with cost $K' > K$. This can only happen if there exists at least one job j that was non-tardy in the fractional solution and after the rewrite rule became tardy in the integer solution.

Since we know that at least one job has become tardy after the rewriting, assume without loss of generality that job j_2 is the first job that has become tardy due to the rewrite rule. Furthermore let job j_1 be the job that is placed directly before job j_2 in the rewritten schedule. Now inspect why job j_2 is placed after job j_1:

- job j_2 was not available at time $C_{j_1} - p$ and $d_{j_2} < C_{j_1} + p$. This cannot happen because then job j_2 would fulfill the requirements mentioned in Step 4e of the rewrite rule and thus it would have been placed immediately as it got released (and thus before job j_1).
- job j_2 was available at time $C_{j_1} - p$. The fact that job j_1 is placed before job j_2 means that $d_{j_1} \leq d_{j_2}$ and thus job j_1 and j_2 cannot be switched. Furthermore all jobs that are placed before job j_1 will also have smaller due dates, thus switching with these jobs is not beneficial either. Using Lemma 4 we know that by relaxing the problem the number of completely processed jobs till a given time cannot increase. This means that, since job j_2 cannot be on-time in the integral case without making another job tardy, there could not have been enough of capacity beforehand in the fractional solution to have all jobs on time, thus contradicting our assumption that everything was on-time in the fractional solution.

□

In case there are tardy fractional parts, it is not known beforehand which jobs will become tardy in the integral solution and thus Step 4e in the rewrite rule is not possible. Hence, this rewrite rule cannot be used when there are tardy fractional parts.

5 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) holds also 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 a job j to time t is:

$$c_{jt} = \begin{cases} w_j \text{ if } t > d_j \\ 0 \quad \text{otherwise.} \end{cases}$$

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 make use of the same approach as in Section 4: we assume two jobs j_1 and j_2 to be non-nested (i.e. job j_1 is nested in job j_2 and vice versa). The proofs for Lemma 6 and Lemma 7 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 exchanging between the last two 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 Figure 4 in Section 4. An exchange between the last two intervals will increase the cost because job j_1 will still be tardy and job j_2 will become tardy after the exchange. An exchange between the middle two intervals will also increase the cost since job j_1 will become tardy while job 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_1 and j_2 are complicating jobs if their due dates and release dates satisfy the following condition:

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

From the 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 with $d_j - r_j < 2p$). If such a job j existed and was partially assigned to an on-time interval and a tardy interval, then job j would be always selected for possible branching with another job j', where j and 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 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 will be on-time
– A child node where job j will be tardy.

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 to time t is:

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

Since this objective in its most extreme form (when $C_j > d_j + p$ for a job j) behaves similarly to the weighted number of late jobs (cost will not increase anymore by placing job 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 and the counterexample is the same as the one given for the sum of weighted late jobs, depicted in Figure 4. 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 rewritten to an integral solution of same cost. This means that the cases where such an equal ordering exists are polynomial solvable.

6 Column generation

A big disadvantage of the used time indexed formulation is that it uses huge amounts of memory. For the biggest of the problems we tested this 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.

We divide the complete time horizon into a number of time frames. This idea of dividing the complete time horizon is independently suggested by Bigras, Gamache, and Savardan (2005) for solving the $1||w_jT_j$ problem by means of column generation. All time frames combined should cover the complete time horizon completely, and consecutive time frames may overlap maximally $p-1$. Each time frame b has a start time μ_b and end time ω_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. $\mu_{b+1}+p-1=\omega_b$). Now we need to find a suitable time frame plan for each of the time frames such that the total idle time in the overlap is equal to or greater than $p-1$, ensuring that we do not place more jobs than possible in the overlap. Furthermore, we need to make sure that all jobs are processed in exactly one of the selected time frame plans.

Now the complete ILP model is as follows:

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

subject to

$$\sum_{i=1}^{m} y_{ji} x_i = 1 \qquad j = 1 \dots n \tag{4}$$

$$\sum_{i=1}^{m} q_{bi} x_i = 1 \qquad b = 1 \dots B \tag{5}$$

$$\sum_{i=1}^{m} z_i q_{bi} x_i + \sum_{i=1}^{m} y_i q_{b+1,i} x_i \geq p-1 \qquad b = 1 \dots B-1 \tag{6}$$

$$x_i \in \{0,1\} \ \forall i, \tag{7}$$

where we have the decision variables

$$x_i = \begin{cases} 1 \text{ if time frame plan } i \text{ is selected} \\ 0 \text{ otherwise,} \end{cases} \tag{8}$$

the indicators

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

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

c_i denotes the cost of time frame plan i, y_i denotes the idle time in the front overlap of time frame plan i, and z_i denotes the idle time at the end overlap of time frame plan i. Constraint (4) ensures that all jobs will be present in exactly one selected time frame plan and Constraint (5) ensures that for each time frame exactly one time frame plan will be selected. Finally Constraint (6) 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 (7) to $x_i \geq 0$. The resulting LP-relaxation we will solve with column generation.
For each type of constraint in the master problem we get a dual multiplier:

- From (4) we get a dual multiplier π_j for each job j.
- From (5) we get a dual multiplier τ_b.
- From (6) we get a dual multiplier ϕ_b for each overlap between time frame b and $b+1$.

For the pricing problem we need to solve a similar problem as the original one for each of the time frames we introduced. 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:
  - $A_j^b = \{a \in A_j | \mu_b + p \leq a \leq \omega_b\}\ \forall j$ denoting the possible completion times of job 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 that 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 α_b and β_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 the α job starts before the start time of the time frame and job β ends after the end time of the time frame. For the first time frame job α does not exist and for the last frame job β 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 being completed at time t such that it corresponds exactly to its contribution to the reduced cost. If a job 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 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 α_b and β_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

$$\sum_{t \in A_j^b} x_{j,t} \leq 1 \text{ for all } j \in N \tag{9}$$

$$\sum_{t \in A_\alpha^b} x_{\alpha_b,t} = 1 \tag{10}$$

$$\sum_{t \in A_\beta^b} x_{\beta_b,t} = 1 \tag{11}$$

$$\sum_{j=1}^{n} \sum_{t' \in H_t^b \cap A_j^b} x_{j,t'} + \sum_{t' \in H_t^b \cap A_\alpha^b} x_{\alpha_b,t'} + \sum_{t' \in H_t^b \cap A_\beta^b} x_{\beta_b,t'} \leq 1 \text{ for all } t \in K^b \cup G^b \quad (12)$$

$$x_{j,t} \in \{0,1\}, \text{for } j \in N \cup \{\alpha, \beta\}, t \in A_j^b \quad (13)$$

where Constraint (9) ensures that all regular jobs are processed at most once and Constraint (10) and Constraint (11) ensure that the alpha and beta job are processed exactly once. Finally Constraint (12) 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 respectively, can be set according to the assignment of the α and β 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 parallelize the solving of the subproblems without any problem. 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.

7 Experimentational Results

For testing the time-indexed formulation, the branching rule, and the rewrite rule 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 1. 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 rewrite rule 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

Jobs	P	Average Time for solving (ms) Cplex	Average Time for solving (ms) B.R.	Average Number of Nodes Cplex	Average Number of Nodes B.R.	Average Number of iterations Cplex	Average Number of iterations B.R.	Average Integrality Gap (‰)
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 rewrite rule
= : Integrality gap was 0 for all instances.

Table 1. Results weighted tardiness

Jobs	P	Average Time for solving (ms) Cplex	Average Time for solving (ms) B.R.	Average Number of Nodes Cplex	Average Number of Nodes B.R.	Average Number of iterations Cplex	Average Number of iterations B.R.	Average Integrality Gap (‰)
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 rewrite rule
*Cplex could not solve on or more instances.

Table 2. Results weighted late jobs

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 non-existent, 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.

If we look at the first line (70 jobs with $P = 5$) in Table 1 we see that with the branching rule and rewrite rule 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 rewrite rule 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 rewrite rule could convert the found fractional solution to an integral solution of same cost. In the cases the rewrite rule 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 was always more than or equal to the number of nodes needed with the branching rule and rewrite rule.

The results for the objective functions weighted late jobs and weighted late work can be found in Table 2 and Table 3 respectively.

		Average Time for solving (ms)		Average Number of Nodes		Average Number of iterations		Average Integrality
Jobs	P	Cplex	B.R.	Cplex	B.R.	Cplex	B.R.	Gap (‰)
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 rewrite rule

Table 3. Results weighted late work

If we look at these two tables 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 Section 5 these instances could be solved.

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

8 Conclusion and further research

We have looked at the $1|r_j, p_j = p|\sum w_j T_j$ problem. We showed that in the case all jobs have a common due date the problem is solvable in polynomial time. Furthermore we showed that the problem where all jobs share a common weight or a common release date can also be solved in polynomial time. For the general problem we showed that when there exists no pair of complicating jobs (i.e. two jobs j_1 and j_2 for which $r_{j_1} > r_{j_2}$, $d_{j_1} < d_{j_2}$, and $w_{j_1} < w_{j_2}$ holds), the problem $1|r_j, p_j = p|\sum w_j T_j$ can be solved in polynomial time. The common due date, common weight, and common release date problems are special versions of the general problem where we can always ensure that no pair of complicating jobs exists.

For the general case of arbitrary due dates and the possible existence of a pair of jobs for which the condition holds, we give a branching rule that can be used. This branching rule exploits the specific properties of the problem. Furthermore a rewrite rule has been formulated to convert certain fractional solutions into integer ones.

We implemented the presented branching rule and rewrite rule in a Java program that made use of Cplex to solve the ILPs. By making use of the extra knowledge of the problem we were able to almost always improve on the number of iterations and nodes needed to solve instances of the problem.

We also showed that the same approach holds for the total weighted late work problem and for the total weighted late jobs problem. For these two problems we only need to change the definition of complicating jobs a bit. Furthermore the same branching rule works very well for the total weighted late work problem and with a small enhancement also works really well for the total weighted late jobs.

We furthermore discovered that everything for the single machine case also works for the m identical, parallel machine problem, since going from a single machine to m machines only requires a change in the right hand side of a subset of the constraints from 1 to m, while the constraint matrix itself stays exactly the same.

Although we have given a good characterization of the problem and the suggested branching rule performs really well, it is still an open question whether the general problem $1|r_j, p_j = p|\sum w_j T_j$ is polynomial solvable.

References

Akturk, M. and 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., and 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., and Savardan, G. (2005). Time-indexed formulations and the total weighted tardiness problem, *Technical report*, GERAD and Ecole Polytechnique de Montreal.

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

Graham, R., Lawler, E., Lenstra, J., and Rinnooy Kan, A. (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. (1977). A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness, *Annals of Discrete Mathematics* **1**: 331–342.

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

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

Nemhauser, G. and Wolsey, L. (1988). *Integer and Combinatorial Optimization*, Wiley-Interscience Series in Discrete Mathematics and Optimization, Wiley.

Roos, C. (2005). Private communication.

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