Scheduling with tails and deadlines

Francis Sourd
LIP6, Université Pierre et Marie Curie, Paris
Francis.Sourd@lip6.fr

Wim Nuijten
ILOG S.A., 9 rue de Verdun, 94253 Gentilly Cedex
nuijten@ilog.fr

Abstract

This paper discusses scheduling problems of operations with tails. While tails are usually used in the literature to model due dates or deadlines, we show it may be interesting to consider tails and deadlines as two different things, especially in shop problems. Then, we review classical one machine and parallel machine problems to show which problems can be still solved in polynomial time in presence of tails and deadlines. We show that both deadlines and tails can efficiently be modeled by a minimax objective function f_{max}. In this way, several problems can be solved in quadratic time but, by considering the specific properties of tails and deadlines and introducing specific data structures, we also show that these problems can be solved in $O(n \log n)$ time. We also show that $P|p_j = 1, r_j|f_{max}$ can be solved in $O(n^2)$ time.

Keywords : scheduling, release dates, deadlines, due dates, tails, minimax objective function, shop scheduling problem, lower bound.

Introduction

Literature distinguishes deadlines from due dates by the fact that a deadline must absolutely be met [Pin95]. Hence due dates are generally associated with optimization criteria such as lateness, tardiness or number of late jobs while deadlines are problem constraints. To our knowledge, only little research effort was devoted to problems in which the operations have a due date and a deadline that is they may be completed after their due date, but not after their deadline. In [HP94], the minimization of the weighted number of tardy jobs on a single machine is studied but this problem is NP-complete even without deadlines. In [GPW97], polynomial algorithms for some preemptive variants of this problem are given. Moreover, in the latter problem, some operations have a deadline while others have a due date. Thus, giving *both* deadlines and due dates to operations to be scheduled appears to be an original approach, which is shown in §1.3 not to be purely theoretical. It gives powerful properties for shop problems.

For the sake of clear notations, we are not going to speak of due dates anymore. These due dates will be substituted by the concept of tails which is presented in Section 1. Then we review classical one machine (Section 2) and parallel machine (Section 3) problems to show which ones can be solved in polynomial time in presence of tails and deadlines. The different classes of problems will be specified by the well-known $\alpha|\beta|\gamma$-notation [GLLK79].

1 Scheduling with tails

1.1 Notations

In all the problems considered in this paper, a set $\mathcal{O} = \{1, 2, \cdots, n\}$ of n jobs is to be scheduled either on a single machine $\mathcal{M} = \{1\}$ in Section 2 or on a set of m parallel machine $\mathcal{M} = \{1, 2, \cdots, m\}$ in Section 3. A job (i, q_i) is defined as a pair made of an operation (or a task) i and a tail q_i. Each operation i has a processing time p_i. It may also have a release date r_i and may be preemptive or not. When the operation i ends, the job is not completed: some amount of work is necessary to complete the job — for example, transportation to the customer. q_i estimates the duration of this work. Hence, e_i denotes the end time of operation i while $C_i = e_i + q_i$ denotes the completion time of job (i, q_i). When no confusion is possible, the job (i, q_i) will be simply denoted by i.

The relationship between due dates and tails is well known. Let us consider a problem in which each operation has no tail but a due date d_i. Its lateness is by definition $L_i = e_i - d_i$. Then, if we set $q_i = -d_i$, the lateness of the operation i is equal to the completion time of job (i, q_i). As a consequence, the problems $\alpha|\beta|L_{max}$ and $\alpha|\beta, q_j|C_{max}$ are equivalent. In this paper, we are going to introduce a deadline d_i for each operation i. So in order to avoid confusion with due dates, we will consider only tailed jobs.

1.2 Criteria

In this section, we show that, when scheduling with tails, the only useful optimization objective is the makespan. Problems with other criteria can be reduced to *untailed* problems. Therefore, in the rest of the paper, we will consider the minimization of the makespan, except to recall some results of complexity.

1.2.1 Due-date-related criteria

Due dates D_j can be introduced for tailed-jobs. So the lateness of a job j is $L_j = C_j - D_j$ and the maximum lateness is as usual $L_{max} = \max_{j \in \mathcal{O}} L_i$.

Proposition 1. *$\alpha|\beta, q_j|L_{max}$ and $\alpha|\beta, q_j|C_{max}$ are equivalent.*

Proof. We have already shown that $\alpha|\beta, q_j|C_{max}$ is equivalent to $\alpha|\beta|L_{max}$. Let us consider a feasible schedule of the instance of the problem $\alpha|\beta, q_j|L_{max}$ if which each job has a due date D_j. We now consider an instance of the problem $\alpha|\beta|L_{max}$ in which each operation j is given a due date $D_j - q_j$. The lateness of j is $e_j - (D_j - q_j) = C_j - D_j = L_j$. Therefore $\alpha|\beta, q_j|L_{max}$ and $\alpha|\beta|L_{max}$ are equivalent, which completes the proof. $\square$

For the reasons given in the proof, $\alpha|\beta, q_j|\sum w_i T_i$ and $\alpha|\beta, q_j|\sum w_i U_i$ are equivalent to $\alpha|\beta|\sum w_i T_i$ and $\alpha|\beta|\sum w_i U_i$.

1.2.2 Mean and weighted flow time

The weighted flow time for jobs is defined as $\sum_i w_i C_i = (\sum_i w_i e_i) + (\sum_i w_i q_i)$. Since $\sum_i w_i q_i$ is a constant, $\alpha|\beta, q_j| \sum w_i C_i$ is equivalent to $\alpha|\beta| \sum w_i C_i$. Therefore, we do not give any algorithms for these criteria.

1.3 The importance of considering deadlines

Models with deadlines and tails can be helpful to find properties of scheduling problems with several machines (shop environment). A deadline for an operation on a machine may come from the problem definition or by deductions resulting of the machine saturation [CP89, NtL98]. A tail can represent the duration of a series of operations (on non critical machines) and/or transportation times.

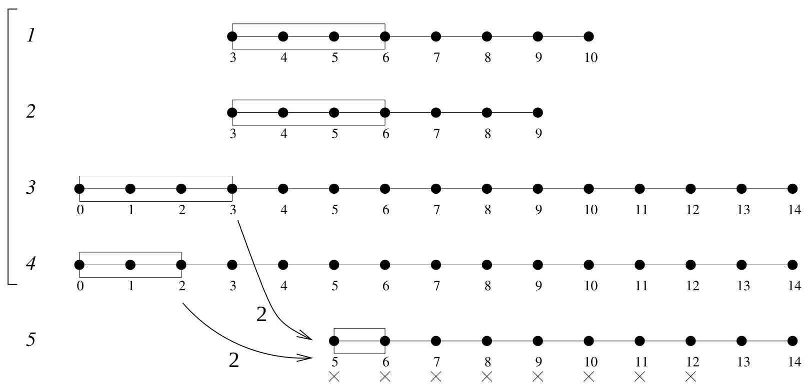

Figure 1: Operations 1, 2, 3 and 4 on the same resource. 2 time units are required between the end of operations 3 and 4 and the start of 5.

For instance, Figure 1 focuses on five operations of a more complicated shop problem. Each line of the figure represents an operation. For example, the first line refers to operation 1. Its release date r_1 is 3, its due date d_1 is 10 and its processing time p_1 is 3. Moreover, we assume that operations 1, 2, 3 and 4 are to be scheduled on a single machine. Finally, it is also assumed that operation 5 must start at least 2 time units after the end of operations 3 and 4. If we set $q_3 = q_4 = 2$, operation 5 must start after $\max(C_3, C_4)$. Therefore, if we set $q_1 = q_2 = -\infty$, the optimal solution of $1|r_j, d_j, q_j|C_{max}$ for the set of operations $\{1, 2, 3, 4\}$ is a lower bound on the start time of operation 5. The reader can easily verify that the minimum makespan (with tails) of this problem is 13.

Unfortunately, $1|r_j, d_j, q_j|C_{max}$ is NP-complete since $1|r_j, d_j|C_{max}$ is NP-complete [GJ77]. For the same reason, $1|r_j, q_j|C_{max}$ is also NP-complete but its preemptive relaxation can be solved in $O(n \log n)$ time [Car82]. Assuming that the deadlines are relaxed in our example, operations with a tail $-\infty$ can be scheduled after all the other operations and the makespan is $\min(r_3, r_4) + p_3 + p_4 + \min(q_3, q_4) = 7$. Thus, the relaxation of deadlines can lead to significant loss of information. The following section presents two single machine problems with tails and deadlines that can be solved in polynomial time.

2 Single machine problems

Jackson's rule [Jac55] that schedules first the operation with the smallest deadline can solve two famous one machine problems with deadlines: the problem without release dates and without preemptions and the problem with release dates and with preemptions. We show that both these problems can be still solved efficiently in presence of tailed operations. The two algorithms are based on a transformation we first present.

2.1 Expressing tails and deadlines with a minimax objective function

We present an objective function f_{max} that models both the tail and the deadline of each operation in $\mathcal{O}$. Let us consider an operation j with a tail q_j and a deadline d_j, we define the function $f_j : \mathbb{R} \longmapsto \mathbb{R}$:

$$t \to f_j(t) = \begin{cases} t + q_j & \text{if } t \leq d_j \\ b & \text{otherwise} \end{cases}$$

b is a "big" problem-dependent constant that can be defined as $\max_{j \in \mathcal{O}}\{d_j + q_j\}$. The objective function f_{max} is then defined for any feasible schedule as $\max_{j \in \mathcal{O}} f_j(e_j)$.

Obviously, an optimal solution of a scheduling problem $\alpha|\beta, d_j, q_j|C_{max}$ is also an optimal solution of the problem $\alpha|\beta|f_{max}$. Conversely, $\alpha|\beta, d_j, q_j|C_{max}$ has no feasible solution if and only if the optimum of $\alpha|\beta|f_{max}$ is b.

2.2 Non-preemptive problem with precedence relations

Lawler developed a simple algorithm to solve in $O(n^2)$ time the problem $1|prec|f_{max}$ when the functions f_j are non-decreasing [Law73]. These conditions are clearly satisfied by the functions defined in § 2.1. Since the functions are non-decreasing and there is no release date, there exists an optimal solution without idle time. In Lawler's algorithm, the sequence of operations is constructed in reverse order. The end time of the schedule is clearly $t = \sum_j p_j$ (cf Figure 2). At each step of the algorithm, the — unscheduled — operation $i^\star$ that minimizes $f_i(t)$ is scheduled between dates $t - p_{i^\star}$ and t is set to $t - p_{i^\star}$. This is repeated until t becomes 0.

Figure 2 shows how f_1, f_2, f_3 and f_4 can be defined to schedule without preemption (and without r_j) the four jobs defined in Figure 1. The latest end time of operations is $e_{max} = \sum_{j=1}^{4} p_j = 11$, the makespan is $\max_{1 \leq j \leq 4} f_j(e_j) = e_3 + q_3 = 13$. We can notice that the produced schedule does not satisfy release date constraints.

If we consider the problem without precedence constraints, we can use the properties of our f_j functions to improve the complexity of the problem:

Theorem 2. *$1|d_j, q_j|C_{max}$ can be solved in $O(n \log n)$ time.*

Proof. We introduce a heap $\mathcal{A}$ to store the operations available at date t and to find the operation with the smallest tail.

$t \leftarrow \sum_{i \in \mathcal{O}} p_i$
$\mathcal{A} \leftarrow \emptyset$
while $t > 0$ **do**
  for each $k \notin \mathcal{A}$ such that $d_k \geq t$ **do** $\mathcal{A} \leftarrow \mathcal{A} \cup \{k\}$

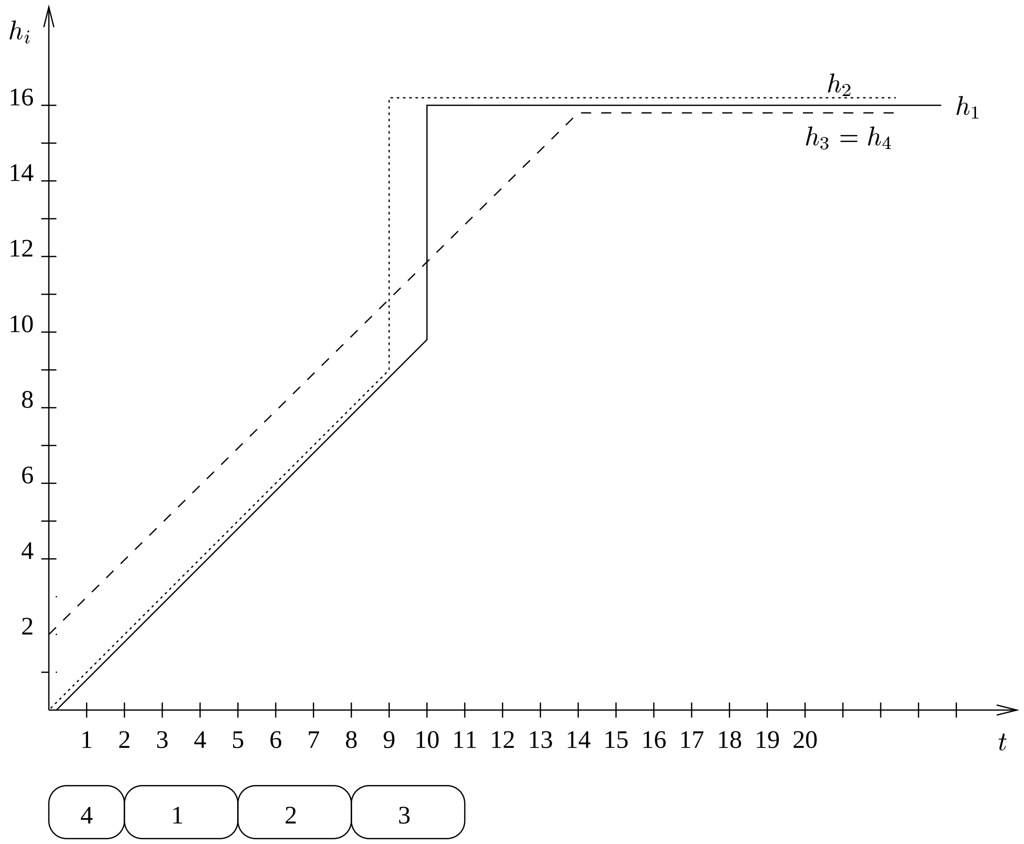

Figure 2: f_1, f_2, f_3, f_4 and an optimal schedule for operations in Figure 1

let $i^\star$ be the operation with the smallest tail in $\mathcal{A}$
$\mathcal{A} \leftarrow \mathcal{A} - \{i^\star\}$
$i^\star$ scheduled between $t - p_{i^\star}$ and t
$t \leftarrow t - p_{i^\star}$
end

Each operation k is inserted only once into $\mathcal{A}$ and is also deleted exactly once. Both operations require $O(\log n)$ time so that the time complexity of this algorithm is $O(n \log n)$. □

This algorithm can be adapted in presence of a precedence graph $G(\mathcal{O}, E)$. At each step, the operation k must be inserted into $\mathcal{A}$ only if all its successors are scheduled. Because of the verification, the overall complexity of the algorithm is $O((n+|E|)\log n)$. Moreover, if we assume that the functions f_i are continuous and if there are few intersections between them, it may be interesting to maintain dynamically the sorted list of the values $f_1(t), f_2(t), \cdots, f_n(t)$, which can be done with usual techniques of computational geometry. Hence, the complexity of $1|prec|f_{max}$ is $O((n+|E|+a)\log n)$ where a is the number of intersections between the f_i functions.

2.3 The preemptive problem with release dates

In most scheduling problems, release dates cannot be relaxed. Therefore, we are now interested in the preemptive problem with release dates, tails and deadlines. We first present an algorithm due to [BLLK83] that solves $1|prmp, prec, r_j|f_{max}$ in $\mathcal{O}(n^2)$ time. It will be called BLLRK in this paper. The transformation in §2.1 can also be used to solve our problem with tails and deadlines but we show that the problem $1|prmp, r_j, d_j, q_j|C_{max}$ is solved in $O(n \log n)$ time.

The execution of the algorithm of BLLRK for an instance of 6 operations is illustrated in Figure 3. This algorithm first schedules, without preemptions, the operations in the order of their release dates r_i (step 1). This schedule, that we will call *the block schedule*, consists of different blocks of operations (a block is a maximum set of operations scheduled consecutively without idle time). The next step of the algorithm reorganizes the execution of operations within each block (with preemption allowed). Assuming that two blocks are identical if and only if they have identical start time and end time and contain the same operations, the following lemma is satisfied:

Lemma 3 ([BLLK83]). *There exists an optimal solution of* $1|prmp, prec, r_j|f_{max}$ *whose blocks are identical to the blocks of the block-schedule.*

Each block B of the block-schedule is rearranged as follows:

- if the block contains only one operation, it is not rearranged;
- let s and t be respectively the start and end time of B and let $i^\star$ be an operation in B, with no successor in B, that minimizes $f_i(t)$.
  - use a recursive call to find an optimaol schedule $S_{i^\star}$ for the instance restricted to the operations in $B \backslash \{i^\star\}$;
  - schedule $i^\star$ within the idle periods of $S_{i^\star}$.

i	1	2	3	4	5	6
r_i	0	1	5	6	15	17
p_i	6	2	3	2	3	1
d_i	16	17	11	11	25	23
q_i	8	10	7	12	1	2

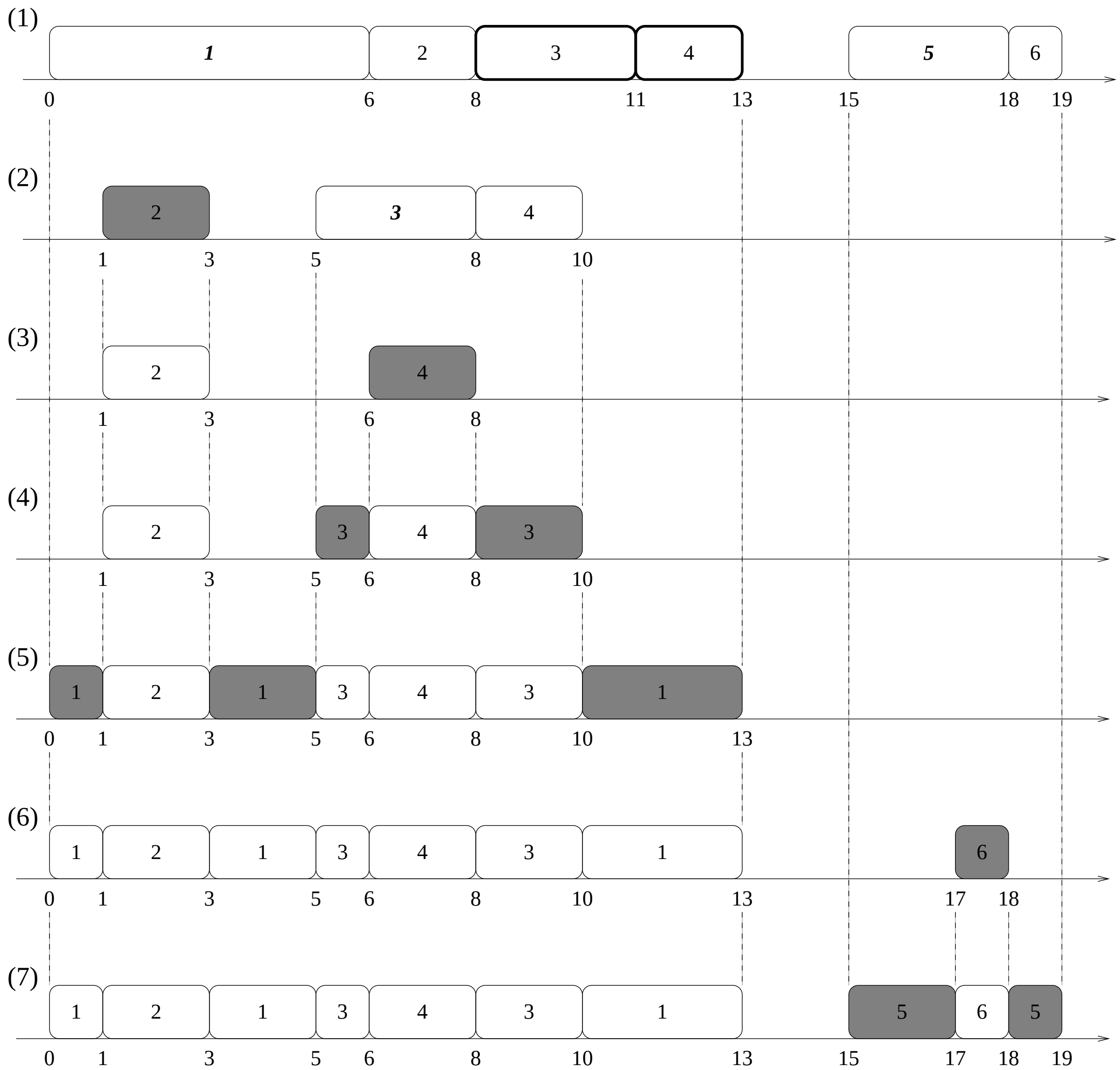

Figure 3: An execution of BLLRK

In order to solve a problem with n operations, the block schedule is constructed in linear time and, for each block B (with n_B operations), the operation $i^\star$ is scheduled in $O(n_B)$ time once the idle intervals are computed by the recursive call. As a consequence, the time complexity of the algorithm is $O(n^2)$. It is also easy to prove that the solution found by this algorithm has at most $n-1$ preemptions. The optimality of the algorithm is proved by observing that for each block that ends at date t, $f_{i^\star}(t)$ is a lower bound for the minimum f_{max}. In the recurrence, only sub-problems of the initial problem are considered so that all the $f_{i^\star}(t)$ values computed at each step are lower bounds. Since the constructed schedule is feasible, the maximum $f_{i^\star}(t)$ value found during the algorithm is the optimal solution value.

Exploiting the special properties of the f_i functions and using more elaborated structures, we created an algorithm — called SN — that improves the complexity of the problem with tails and deadlines.

Theorem 4. *The algorithm* SN *solves* $1|prmp, r_j, d_j, q_j|C_{max}$ *in* $O(n \log n)$ *time.*

The description of SN requires additional notations that we introduce by illustrating its execution on one instance (Figure 4). First of all, we assume without loss of generality that the operations are sorted in non-decreasing order of their release dates. Hence, $i \leq j$ implies $r_i \leq r_j$. If two operations i and j are scheduled in the same block B of the block schedule, any operation k such that $i \leq k < j$ is in B. For example, in Figure 3-(1), the two blocks are $\{1, 2, 3, 4\}$ and $\{5, 6\}$. In the same way as BLLRK , the blocks of the schedule provided by SN will be the same as the blocks of the block schedule.

The blocks of the block schedule are reorganized successively in reverse order and the operations in each block are also scheduled in reverse order. Figure 4 shows how the block $\{1, 2, 3, 4\}$ (see Figure 3) is scheduled by SN . The block $\{5, 6\}$ is assumed to have already been scheduled by SN . Let $\mathcal{S}^-$ be the block schedule of the unscheduled operations and let T be the end time of $\mathcal{S}^-$. Note that $\mathcal{S}^-$ may violate the deadlines constraints. In the example, Fig. 3(1) represents $\mathcal{S}^-$, $T = 13$ and operation 4 ends after its deadline $d_4 = 11$. p_i^+ is the processed time of i after T — *ie* the part of i that has already been scheduled — and $p_i^- = p_i - p_i^+$ denotes the processing time of i before T — *ie* the part of i that has not been scheduled yet. Let f be the operation with the greatest release date such that $r_f + \sum_{i \geq f} p_i^- = T$. Such a f exists since the first operation of the last block of $\mathcal{S}^-$ satisfies this equality. Obviously, all the unscheduled parts of the operations $i \geq f$ must be scheduled between r_f and T and no other operation can be scheduled in this time interval. The set of operations $\mathcal{B}_f = \{i \geq f \,|p_i^- > 0\}$ will be called the *current block*. In the example, at $T = 13$, $f = 1$ and the current block is $\{1, 2, 3, 4\}$. Then SN selects the operation $i^\star \in \mathcal{B}_f$ with the smallest tail among the available operations $\mathcal{A} = \{i \,|\, r_i < T \leq d_i\} = 1, 2$. It finds $i^\star = 1$. Finally, the algorithm determines which "length" of $i^\star$ must be scheduled. We can indeed notice that if 4 time units of operation 1 are scheduled in time interval $[9, 13]$, operations 3 and 4 cannot be scheduled any more. In order to avoid to reach such a deadlock, we will introduce for each operation $j \in \mathcal{B}_f$ the value $\Delta_{fj} = r_f + \sum_{f \leq k < j} p_k^- - r_j$, which is the difference between the start time of j in $\mathcal{S}^-$ and its release date. In Fig. 4-(2), we can observe that each time one time unit of $i^\star$ is scheduled, the Δ_{fj}-values are decreased by one for all $j > i^\star$. Therefore, in Fig. 4-(3), once 3 time units of operation 1 are scheduled, $\Delta_{13} = 0$. It means that $r_1 + p_1^- + p_2^- = r_3$ and $r_3 + p_3^- + p_4^- = T$. So $\{3, 4\}$ becomes the current block (Fig. 4-(4)).

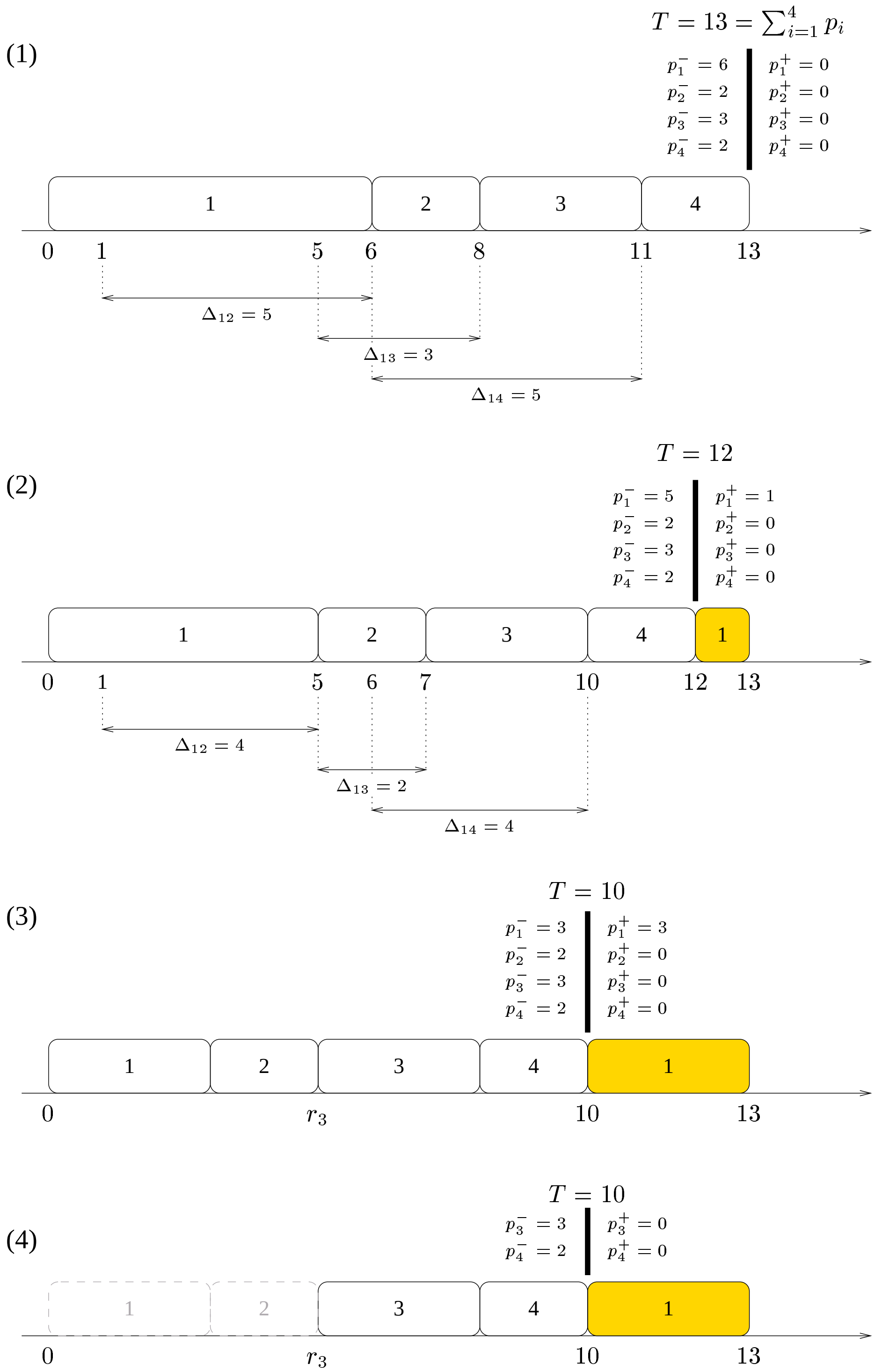

Figure 4: The first steps of SN

The scheduling rule of SN may be formulated as follows : Schedule in reverse order an available job of the current block which has the smallest tail until one of the two events is met :

1. the operation is completely scheduled ;
2. one of the Δ_{fj} becomes 0.

We now give a more formal description of SN . As in the algorithm for the f_{max} objective function, the block schedule is first constructed. Then the procedure `schedule_block` is called for each block B of the block schedule in decreasing order of their start time. f is the operation with the smallest release date in the block ($B = \mathcal{B}_f$). The end time T of B is equal to $r_f + \sum_{i \in \mathcal{B}} p_i^-$.

procedure `schedule_block`(f, T)
begin
  if $T > r_f$ **then**
  begin
    for each $k \notin \mathcal{A}$ and $d_k \geq T$ **do** $\mathcal{A} \leftarrow \mathcal{A} \cup \{k\}$
    $\mathcal{Q}(f, \mathcal{A}) \leftarrow \{i \,|\, i \in \mathcal{A} \text{ and } r_f \leq r_i\}$
    if $\mathcal{Q}(f, \mathcal{A})$ is empty **then** there is no feasible schedule **else**
    let $i^\star$ be an operation of $\mathcal{Q}(f, \mathcal{A})$ with a minimum tail
    let $j^\star$ be an operation that minimizes $\{\Delta_{fj} \,|\, p_j^- > 0 \text{ and } r_{i^\star} < r_j\}$
    $t \leftarrow \max\{T - p_{i^\star}^-, T - \Delta_{fj^\star}\}$
    let $i^\star$ be the operation scheduled in time interval $[t, T[$
    `decrease`$(i^\star, T - t)$
    case t **of**
      $T - p_{i^\star}^-$: $\mathcal{A} \leftarrow \mathcal{A} - \{i^\star\}$
          `schedule_block`(f, t)
      $T - \Delta_{fj^\star}$: `schedule_block`$(j^\star, t)$
          `schedule_block`$(f, r_{j^\star})$
    end
  endif
end

The procedure `decrease`(i, δ) will be described later. It decreases the processing time of i by δ (*ie :* $p_i^- \leftarrow p_i^- - \delta$). Then it updates the data structure to calculate the values Δ_{ij}.

The complete proof of the validity and of the time complexity of Theorem 4 is quite long because it requires the introduction of specific data structures. For this reason, it is given in appendix at the end of the paper. Here we only present a sketch of this proof.

The proof of validity is based upon the fact that `schedule_block` provides a feasible schedule whose makespan is a lower bound for the problem. The constructed schedule is shown to have at most $n - 1$ preemptions. Two data structures are then presented to compute $i^\star$ and $j^\star$ in $O(\log n)$ time. Both these structures are also updated in $O(\log n)$ time. That eventually proves that `schedule_block` solves the problem in $O(n \log n)$ time.

We cannot use the traditional heap data structure to find $i^\star$ and $j^\star$ because of the additional constraints "$r_f \leq r_i$" and "$r_{i^\star} < r_j$". Moreover the Δ_{fj}-values are not constant.

At last, one can observe that SN does not always produce the same schedule than BLLRK . For instance, the operations

i	r_i	d_i	p_i	q_i
1	0	3	1	1
2	2	3	1	4
3	0	4	2	2

yield two different schedules with makespan 7.

2.4 Preemptive or non-preemptive relaxation

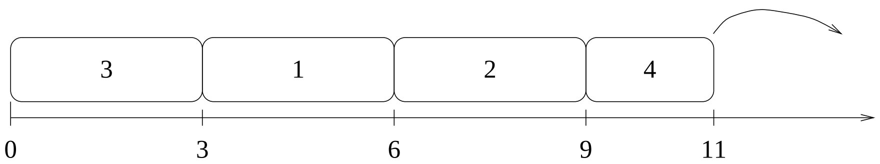

Figure 5: An optimal schedule for the non-preemptive problem with release dates.

The polynomial algorithms presented in §2.2 and §2.3 offer two means for calculating a lower bound for the earliest start time of operation 5 in Figure 1. First, we can relax the release dates by setting r_3 and r_4 to 0 in order to deal with an instance of $1|d_j, q_j|C_{max}$. Otherwise, we can consider the preemptive relaxation which leads to an instance of $1|prmp, r_j, d_j, q_j|C_{max}$. We can remark that the optimal makespan for both these problem is 13. 13 is also the makespan of the corresponding instance of $1|r_j, d_j, q_j|C_{max}$ as shown by Figure 5.

What is the best relaxation to choose? It is not difficult to see that when all release dates are equal, the block-schedule of operations in $B - \{i^\star\}$ in §2.3 has only one block. As a consequence, $i^\star$ is not preempted and the constructed schedule has no preemptions. So, we can conclude that the algorithm presented in §2.3 always finds a greater makespan and then should be prefered to update the earliest start times.

2.5 Flow-time and deadlines

It is well known that $1|prmp, r_j|\sum C_j$ is polynomial [Bak74] but this problem becomes NP-hard as soon as deadlines are added [DL93].

3 Parallel machine problems

In the example presented in Figure 1, the four operations are to be scheduled on a single machine. However, the process presented in §1.3 is still valid when the predecessors of the operation whose earliest start time must be updated are to be scheduled on parallel machines. Hence, in this section, the jobs (i, q_i) are to be scheduled on a machine among a set of m machines $\mathcal{M} = \{1, 2, \cdots, m\}$. As we did for single machine problems, we will specifically be interested in problems with deadlines.

The objective function f_{max} is still very useful to model both tails and deadlines so that we will mainly present simple extensions of existing algorithms for this criterion.

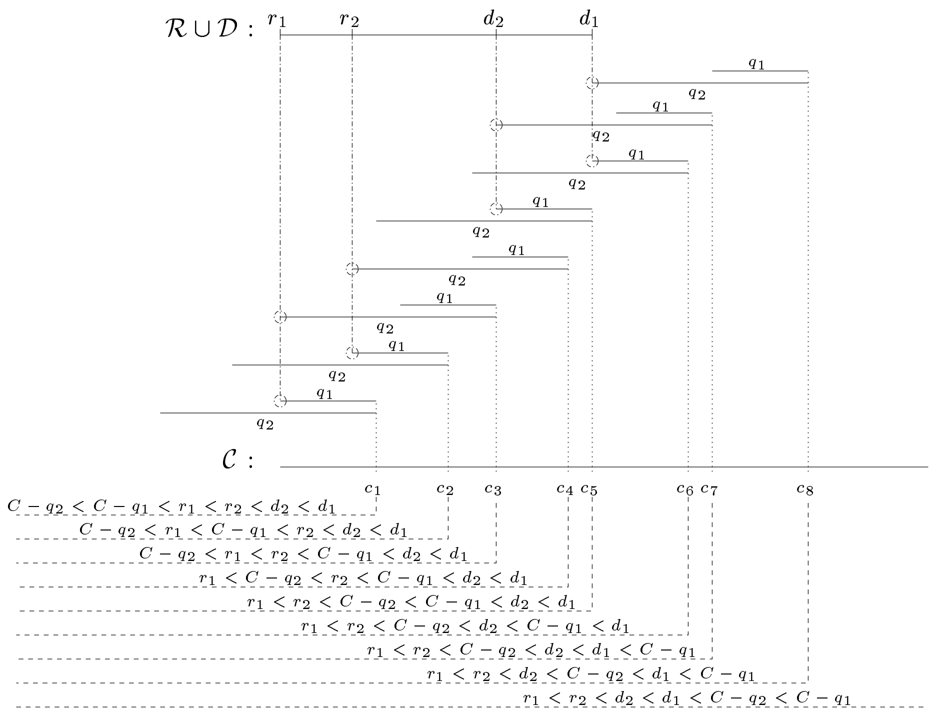

Figure 6: $\mathcal{R}$, $\mathcal{D}$, $\mathcal{Q}$ and $\mathcal{C}$ in the proof of Proposition 6

3.1 Unlimited number of machines

Since it is NP-complete when $m = 1$, the non-preemptive scheduling problem of tasks with release dates and deadlines on parallel machines is NP-complete. Hence, we will be interested in different relaxations of this problem. A common idea when facing a parallel machine environment is to relax the limitation on the number of machines. We have then the (obvious) result :

Proposition 5. *The problem $P\infty|prec, r_j|f_{max}$ can be solved by the critical path method.*

In shop problems, it is well known that if operation i precedes operation j then $r_j \geq r_i + p_i$. These earliest start times are usually updated with Ford-Bellman's algorithm, which is equivalent to the critical path method. As a consequence, if we want to update the earliest start times of an operation o by the process presented in §1.3, the *capacity constraint* on the size of $\mathcal{M}$ must not be relaxed.

3.2 Preemptive relaxation on unrelated machines

Instead of relaxing the capacity constraint, we can relax the non-preemption assumption. We have then a polynomial problem even when the parallel machines are unrelated :

Proposition 6. *The optimal solution of $R|prmp, r_j, d_j, q_j|C_{max}$ can be computed in polynomial time.*

Proof. Let us define the sets $\mathcal{R} = \{r_j \mid j \in \mathcal{O}\}$, $\mathcal{D} = \{d_j \mid j \in \mathcal{O}\}$ and $\mathcal{Q} = \{C_{max} - q_j \mid j \in \mathcal{O}\}$. If we suppose that the set $\mathcal{R} \cup \mathcal{D} \cup \mathcal{Q}$ is totally ordered, that is $\mathcal{R} \cup \mathcal{D} \cup \mathcal{Q} = \{t_1, t_2, \cdots, t_{3n}\}$ with $t_1 \leq t_2 \leq \cdots \leq t_{3n}$, we can use the linear formulation for $R|prmp, r_j, d_j|C_{max}$ of [LL78] to compute the variables $t_{ij}^{(k)}$ and C_{max} where $t_{ij}^{(k)}$ is the processing time of operation i on machine j within the interval $[t_k, t_{k+1}]$. In other words, our problem can be solved in polynomial time once we know the total order on $\mathcal{R} \cup \mathcal{D} \cup \mathcal{Q}$. We are going to show that there are at most $\mathcal{O}(n^2)$ possible total orders on this set (see also Figure 6).

First of all, we know that the sets $\mathcal{R} \cup \mathcal{D}$ and $\mathcal{Q}$ are both totally ordered. Let us consider the set $\mathcal{C} = \{C \mid C = t + q, C_{max} - q \in \mathcal{Q}, t \in \mathcal{R} \cup \mathcal{D}\}$. So

$$|\mathcal{C}| \leq |\mathcal{Q}| \times |\mathcal{R} \cup \mathcal{D}| \in O(n^2)$$

Let $\mathcal{C} = \{c_1, c_2, \ldots, c_{|\mathcal{C}|}\}$ with $c_1 \leq c_2 \leq \cdots \leq c_{|\mathcal{C}|}$. Let $c_0 = -\infty$ and $c_{|\mathcal{C}|+1} = +\infty$. For any $k \leq |\mathcal{C}|$, the condition $c_k \leq C_{max} \leq c_{k+1}$ implies that the set $\mathcal{R} \cup \mathcal{D} \cup \mathcal{Q}$ is totally ordered. In consequence, we have at most $|\mathcal{C}| + 2$ possible total orders on $\mathcal{R} \cup \mathcal{D} \cup \mathcal{Q}$. Using binary search on the values in $\mathcal{C}$, the minimum C_{max} which gives a feasible schedule can then be found by solving at most $O(\log n)$ linear programs. $\square$

This result is more a theoretical complexity result than a usable algorithm to solve the problem. In particular, it may not be used for deriving lower bounds for a shop problem in a branch and bound scheme. To our knowledge, there is no fast algorithm to solve this problem, even if the parallel machines are identical or uniform — $Q|prmp, r_i, d_i|-$ can be solved in $O(mn^3)$ time by reduction to a network flow problem yet [FG86]. For this reason, it may be of practical interest to find fast algorithms to compute lower bounds for these problems as [CP98] did for the problem without deadlines $P|prmp, r_i, q_i|C_{max}$.

However, special cases of the scheduling problem with tails and deadlines on parallel machines can be efficiently solved. In particular, the next section considers that all the operations have unit processing time ($p_i = 1$).

3.3 Unit Execution Time (UET) operations

We will once more consider the objective function f_{max} defined in §2.1 that can model both tails and deadlines. [GLLK79] have shown that $Q|p_i = 1|f_{max}$ can be solved in $O(n^2)$ time by adapting Lawler's algorithm (§2.2). Let s_j be the speed of machine j and let us consider the time intervals $I(k, j) = [k/s_j; (k+1)/s_j]$ for any $k \in \mathbb{N}$ and any $j \in \mathcal{M}$. $I(k, j)$ will contain the k^{th} operation scheduled on machine j. Obviously, there exists an optimal schedule contained in the n intervals $I(k, j)$ with the n smallest end times $(k+1)/s_j$. As for Lawler's algorithm, the n operations are affected to these n intervals in reversed order: at each step, the free interval $I(k, j)$ with the greatest end time $(k+1)/s_j$ is allocated to the non-assigned operation $i^\star$ that minimizes $f_i((k+1)/s_j)$.

In presence of release dates, the problem becomes more difficult because all the operations cannot be scheduled in the n first time intervals. However, a variant of the algorithm presented in §2.3 solves the problem with identical machines $P|p_j = 1, r_j|f_{max}$.

We now present this new algorithm. The m identical machines are numbered from 1 to m. $I(k, j)$ is the time interval (*slot*) $[k, k+1[$ of machine j. These slots are sorted in lexicographical

i	1	2	3	4	5	6	7	8	9	10	11	12	13
r_i	1	3	3	3	3	3	4	4	5	6	6	6	7

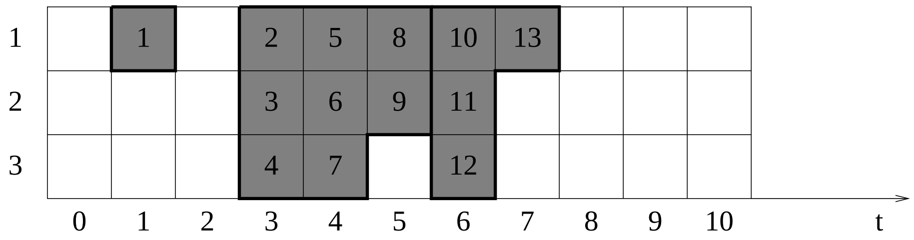

Figure 7: Block-schedule of 13 UET operations on 3 machines

order, that is the successor of $I(k, j)$ is:

$$\begin{cases} I(k, j+1) & \text{if } j \neq m \\ I(k+1, 1) & \text{if } j = m \end{cases}$$

For any slot I, there exist two integers k and j such that $I = I(k, j)$. k and j are respectively the *date* and the *machine* of slot I. The n operations are assumed to be sorted in the order of their release dates. The following algorithm constructs a feasible (non optimal) schedule to create blocks of operations, that will play the same role as the block-schedule in §2.3. In this parallel machine problem, a block is defined as a maximum set of operations scheduled without idle time in consecutive slots.

procedure `create_blocks`($\{1, \cdots, n\}$)
begin
  $I \leftarrow I(-1, 1)$
  $b \leftarrow 0$
  for $i \leftarrow 1$ **to** n **do**
  begin
    if $r_i >$ date of I **then** $I \leftarrow I(r_i, 1)$
      $b \leftarrow b + 1$
      $B_b \leftarrow \emptyset$
    $B_b \leftarrow B_b \cup \{i\}$
    $I \leftarrow$ successor of I
  end
end

The schedule produced by `create_blocks` is the *block schedule.* Each block B_b is a set of operations to which we can associate a start slot and an end slot: each operation in B_b is scheduled in one slot that is between the start slot and the end slot. Figure 7 presents a block schedule of 13 UET operations on 3 machines. This schedule has three blocks. The first block as only one operation $\{1\}$. Its start and end slots are $I(1, 1)$. The second block $\{2, \cdots, 9\}$ starts at $I(3, 1)$ and ends at $I(5, 2)$. The last block $\{10, \cdots, 13\}$ starts at $I(6, 1)$ and ends at $I(7, 1)$. In what follows, two blocks will be stated as identical if they countain the same operations and have identical start and end slots. With these definitions, we can give a variant of Lemma 3:

Lemma 7. *There exists an optimal solution of $P|p_j = 1, r_j|f_{max}$ in which the blocks are identical to the blocks of the block-schedule.*

Proof. For an instance of $P|p_j = 1, r_j|f_{max}$, let us consider an optimal schedule $\mathcal{S}_{\text{opt}}$ and the block schedule $\mathcal{S}_{\text{block}}$ and assume that there exists a slot that is idle in one schedule and assigned in the other. Let I be the first of these slots. Since no operation is delayed by `create_block`, I is necessary idle in $\mathcal{S}_{\text{opt}}$ and assigned in $\mathcal{S}_{\text{block}}$. Let t be the date of I. If all the operations scheduled after I in $\mathcal{S}_{\text{opt}}$ have a release date greater than t, the operation j assigned to the slot I in the block schedule would also have a release date greater than t (from the definition of I), which is a contradiction. Therefore, there exists an operation i in $\mathcal{S}_{\text{opt}}$ scheduled after I with $r_i \leq t$. Job i can be moved into idle slot I and the new schedule is also optimal because f_i is non-decreasing. Iterating this process, an optimal schedule with the same blocks as $\mathcal{S}_{\text{block}}$ is found. ☐

We can build an optimal schedule of a set S of UET operations by the following procedure :

- if S has only one operation, schedule this operation at its release date
- otherwise, let B_b be the block(s) computed by the call to the procedure `create_blocks`(S). For each block B_b :
  - let t_b the date of the end slot of B_b
  - let $i^\star \in B_b$ that minimizes $f_i(t_b + 1)$
  - solve by a recursive call the sub-problem with operations in $B_b - \{i^\star\}$
  - schedule $i^\star$ in the idle slot

The proof of this algorithm is identical to the proof of the algorithm in §2.3. We have finally the property :

Proposition 8. *$P|p_i = 1, r_i|f_{max}$ can be solved in $O(n^2)$ time.*

The proof of Theorem 4 can easily be adapted to show that the problem $P|p_i = 1, r_i, d_i, q_i|C_{max}$ can be solved in $O(n \log n)$ time.

Conclusion

In this paper, we have discussed scheduling problems whose operations have both tails and deadlines. We have presented several problems on single or parallel machines that can be solved in polynomial time. We have shown that the objective function f_{max} can model both tails and deadlines but the complexity of the problems can be improved when we consider specific properties of the tails and deadlines. Moreover, we have shown $P|p_i = 1, r_i|f_{max}$, can be solved in quadratic time.

We plan to investigate further how to make the best use of the lower bounds presented in §1.3 in a branch and bound scheme in order to solve shop scheduling problems.

Acknowledgements : The authors are indebted to Philippe Chrétienne for his careful reading of the previous versions of this paper. The work of the first author is partially financed by ILOG S.A., under research contract ILOG/UPCM no. 980220.

A Proof of Theorem 4

First of all, we recall the definition of $\Delta_{fj} = r_f + \sum_{f \le k < j} p_k^- - r_j$ where we assume that the operations are sorted in the non decreasing order of their release dates. Obviously, $\Delta_{fj} = \Delta_{fi} + \Delta_{ij}$ for any i such that $f \le i \le j$. If f is the first operation of a block $\mathcal{B}$ of a block schedule, for any $j \in \mathcal{B}$, $\Delta_{fj} \ge 0$. Conversely, let us consider a set $\mathcal{B}$ of operations and let $f \in \mathcal{B}$ be an operation with the smallest release date. If, for any $j \in \mathcal{B}$, $\Delta_{fj} \ge 0$ then it is easy to see that the block schedule has only one block that starts at r_f and ends at $r_f + \sum_{k \in \mathcal{B}} p_k^-$. From Lemma 3, there is also an optimal schedule of $\mathcal{B}$ included in this time interval.

We are now going to prove that, for any block $\mathcal{B}$ of the block schedule, `schedule_block` returns an optimal schedule of the operations in $\mathcal{B}$.

A.1 Proof of correctness

Let us consider a block $\mathcal{B}$ of the block schedule. Each operation i of $\mathcal{B}$ has a processing time $p_i^- = p_i$ and $\ell = \sum_{i \in \mathcal{B}} p_i$ is the length of $\mathcal{B}$. If $\mathcal{B}$ has only one operation f, this operation is scheduled with no preemption at its release date : this schedule is optimal.

We now consider that $\mathcal{B}$ has more than one operation. Let f be an operation of $\mathcal{B}$ with the smallest release date and let $T = r_f + \ell$ be the end time of $\mathcal{B}$. Let us assume that, for all blocks of length $\ell < L$, `schedule_block`(f, T) produces an optimal schedule included in the time interval $[r_f, T]$ and let us consider a block $\mathcal{B}$ of length $\ell = L$. In what follows, t and $i^\star$ are the values defined in the description of `schedule_block`.

- If $t \ne T$: `schedule_block` schedules the operation $i^\star$ between t and T. From the definition of $i^\star$, $T + q_{i^\star}$ is a lower bound for the makespan of the schedule of $\mathcal{B}$. Let us consider the set of operations $\mathcal{B}'$ that contains the same operations that $\mathcal{B}$ with processing times

$$p'_i = \begin{cases} p_i & \text{if } i \ne i^\star \\ p_{i^\star} - (T - t) & \text{if } i = i^\star \end{cases}$$

  For any feasible schedule of $\mathcal{B}$, we can build a feasible schedule of $\mathcal{B}'$ by replacing $T - t$ slots in which $i^\star$ is scheduled by idle time. So the optimal makespan for $\mathcal{B}'$ is less than the optimal makespan for $\mathcal{B}$. Let us define, for any $j \in \mathcal{B}'$, $\Delta'_{fj} = r_f + \sum_{f \le k < j} p'_k - r_j$. From this definition, if $j \le i^\star$, $\Delta'_{fj} = \Delta_{fj} \ge 0$. If $j > i^\star$, $\Delta'_{fj} = \Delta_{fj} - (T - t)$. Since $T - t \le \Delta_{fj^\star} \le \Delta_{fj}$, $\Delta'_{fj} \ge 0$. So, the optimal schedule of $\mathcal{B}'$ can be included in time interval $[r_f, t]$. Since $t - r_f < L$, `schedule_block` finds an optimal schedule of $\mathcal{B}'$ and finally the produced schedule is optimal.

- if $t = T$: since $p_{i^\star} > 0$, we necessarily have $\Delta_{fj^\star} = 0$ for some $j^\star \in \mathcal{B}$. Clearly, $r_f < r_{j^\star} < T$. Let us define, $\mathcal{B}_1 = \{i \in \mathcal{B} \,|\, r_i < r_{j^\star}\}$ and $\mathcal{B}_2 = \{i \in \mathcal{B} \,|\, r_i \ge r_{j^\star}\}$. $\mathcal{B}_1$ and $\mathcal{B}_2$ make a partition of $\mathcal{B}$. From the definition of $\Delta_{fj^\star}$:

$$\begin{aligned} r_f + \sum_{i \in \mathcal{B}_1} p_i &= r_{j^\star} \\ r_{j^\star} + \sum_{i \in \mathcal{B}_2} p_i &= T \end{aligned}$$

In any optimal schedule of $\mathcal{B}$ included in the time interval $[r_f, T]$, all the operations that are in $\mathcal{B}_2$ must be scheduled in $[r_{j^\star}, T]$. Therefore, all the operations in $\mathcal{B}_1$ must be scheduled in $[r_f, r_{j^\star}]$. Since $\mathcal{B}$ forms a block, $\Delta_{fj} \geq 0$ for all $j \in \mathcal{B}$. So for any $j \in \mathcal{B}_1$, $\Delta fj \geq 0$ and for any $j \in \mathcal{B}_2$, $\Delta_{j^\star j} = \Delta_{fj^\star} + \Delta_{j^\star j} = \Delta_{fj} \geq 0$. Therefore the operations in $\mathcal{B}_1$ and in $\mathcal{B}_2$ form each one block. The makespan of each of these two blocks is of course a lower bound of the makespan of the schedule of $\mathcal{B}$. Since $r_{j^\star} - r_f < L$ and $T - r_{j^\star} < L$, the produced schedule is optimal.

To conclude, `schedule_block` produces an optimal schedule for $1|prmp, r_j, d_j, q_j|C_{max}$.

A.2 Number of preemptions

Lemma 9. *A block $\mathcal{B}$ scheduled by* `schedule_block` *has at most $|\mathcal{B}| - 1$ preemptions.*

Proof. This lemma is obvious if $\mathcal{B}$ has only one operation. Let us suppose the lemma is valid for all $\mathcal{B}$ such that $|\mathcal{B}| < N$ and let us consider a block $\mathcal{B}$ such that $|\mathcal{B}| = N$. When `schedule_block` is called :

- if $t = T - p_{i^\star}$, $i^\star$ is scheduled without premption. Then the block $\mathcal{B} - \{i^\star\}$ is scheduled between r_f and t with at most $(N-1)-1$ preemptions. So the schedule has at most $N-2$ preemptions (and of course at most $N-1$ preemptions).

- if $t = T - \Delta_{fj^\star}$, only a part of $i^\star$ is scheduled, which causes one preemption. Then two non-empty sub-block $\mathcal{B}_1$ and $\mathcal{B}_2$ are scheduled. $|\mathcal{B}_1| < N$ and $|\mathcal{B}_2| < N$ so the constructed schedule has at most $1 + (|\mathcal{B}_1| - 1) + (|\mathcal{B}_2| - 1) = |\mathcal{B}| - 1$ preemptions.

$\square$

This result shows that, when scheduling a block $\mathcal{B}$, `schedule_block` is called $O(|\mathcal{B}|)$ times. We are now going to show that each instruction of `schedule_block` can be executed in $O(\log n)$ time, which will show that each block $\mathcal{B}$ is scheduled in $O(|\mathcal{B}| \log n)$ time.

A.3 Data structures

We present in §A.3 two data strutures for maintaining minimum elements in dynamic sets :

- $i^\star$ is an operation that minimizes $\{q_i \,|\, i \in \mathcal{A} \text{ and } f < i\}$

- $j^\star$ is an operation that minimizes $\{\Delta_{fj} \,|\, j \in \mathcal{B}_f \text{ and } i^\star < j\}$

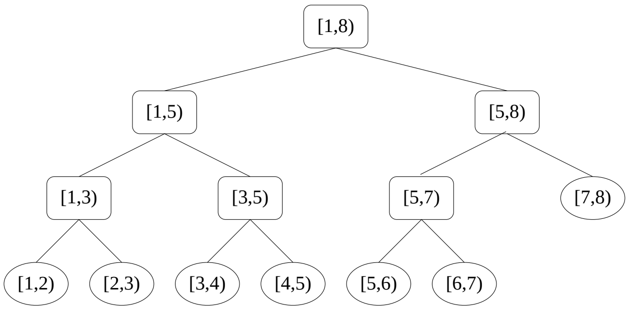

Figure 8: The binary tree $\mathcal{T}$ for a set of 7 operations

Both these structures are based upon the same basic structure, a binary tree $\mathcal{T}$, which is represented in Figure 8 for $n = 7$. This binary tree divides the set of operations into subsets defined as follows. Each node is denoted by a pair of integers $[l, r)$ which means that the subtree of root $[l, r)$ covers the operations $\{l, l + 1, \cdots, r - 1\}$. In this way, the root of $\mathcal{T}$ is denoted by $[1, n + 1)$ and the leaf corresponding to the only operation i is $[i, i + 1)$. The left — resp. right — descendant of node $[l, r)$ is $[l, m)$ — resp. $[m, r)$ — where $m = \lceil \frac{l+r}{2} \rceil$. $\mathcal{T}$ is a binary tree with n leaves so it has at most $2n - 1$ nodes.

We now present how to use $\mathcal{T}$ to query for $i^\star$ and $j^\star$. It is important to remember that the n operations are numbered in the order of their release dates.

A.3.1 Available operations

At each call of the recursive procedure `schedule_block`, the available operation with the shortest tail is searched for ($i^\star$). We cannot use a heap structure to store $\mathcal{A}$ because of the additional constraint that is the operation must be in the current block ($i^\star \geq f$). It is also fruitless to create a heap for each block because sub-blocks are created during the algorithm execution. So we present an original data structure based on $\mathcal{T}$ with the following properties:

- an operation can be inserted or removed in $\log(n)$ time;
- $i^\star$ can be determined in $\log(n)$ time.

At each node $[l, r)$ of $\mathcal{T}$, we associate an operation $\iota[l, r)$ that minimizes $\{q_i \mid i \in \mathcal{A} \text{ and } l \leq i < r\}$. If this set is empty, $\iota[l, r) = 0$. By setting $q_0 = +\infty$, we have the simple relation:

$$q_{\iota[l,r)} = \min\left(q_{\iota[l,m)}, q_{\iota[m,r)}\right) \tag{1}$$

The $\iota[l, r)$-values are initialized after that the block schedule (that completes at time T) is computed. The values of the n leaves are $\iota[i, i + 1) = i$ if $d_i > T$ and $\iota[i, i + 1) = 0$ otherwise. The $O(n)$ inner nodes are initialized in topological order with (1). So, the initialization process is done in linear time. The data structure also implements two update procedures to insert an operation into $\mathcal{A}$ and to remove it from $\mathcal{A}$. They are both based on (1):

procedure `add`$((l, r), i)$;; $\mathcal{A} \leftarrow \mathcal{A} \cup \{i\}$

```
begin
   if l = r − 1 then
      ι[i, i + 1) ← i
   else
      m ← ⌈(l+r)/2⌉
      if i < m then add((l, m), i) else add((m, r), i)
      if q_ι[l,m) ≤ q_ι[m,r) then ι[l, r) ← ι[l, m) else ι[l, r) ← ι[m, r)
   endif
end

procedure remove((l, r), i)  ;; A ← A − {i}
begin
   if l = r − 1 then
      ι[i, i + 1) ← 0
   else
      m ← ⌈(l+r)/2⌉
      if i < m then remove((l, m), i) else remove((m, r), i)
      if q_ι[l,m) ≤ q_ι[m,r) then ι[l, r) ← ι[l, m) else ι[l, r) ← ι[m, r)
   endif
end
```

`add` and `remove` are based on a dichotomic search. Their time complexity is clearly $O(\log n)$. The data structure $\mathcal{T}$ must be able to find an operation $i^\star$ that minimizes $\{q_i \mid i \in \mathcal{A} \text{ and } f \leq i\}$ in $O(\log n)$ time. We can easily verify that the following function performs this task:

```
function i_star((l, r), f)
begin
   if l = r − 1 then return ι[l, r)
   else
      m ← ⌈(l+r)/2⌉
      if f < m then i ← i_star((l, m), f)
                     return i' ∈ {i, ι[m, r)} such that q_i' = min(q_i, q_ι[m,r))
               else  return i_star((m, r), f)
   endif
end
```

A.3.2 Minimum Δ_{fj}

In order to find $j^\star$, we define, for any pair (λ, ρ) of integers such that $1 \leq \lambda < \rho \leq n+1$, $\sigma[\lambda, \rho) = \sum_{\lambda \leq j < \rho} p_j^-$ and $\mu[\lambda, \rho) = \min_{\lambda \leq k < \rho} (\sigma[\lambda, k) - r_k)$. With these definitions, we have $\Delta_{fj^\star} = r_f + \min_{j > i^\star}(\sigma_{fj} - r_j) = r_f + \sigma[f, i^\star) + \mu[i^\star, n+1)$. These values $\sigma[l, r)$ and $\mu[l, r)$ are associated at each node $[l, r)$. We have then the immediate recurrence relations:

$$\sigma[l, r) = \sigma[l, m) + \sigma[m, r) \quad (2)$$

$$\mu[l, r) = \min\left(\mu\left(l, m\right), \sigma[l, m) + \mu\left(m, r\right)\right) \quad (3)$$

Once again, we can initialize the $\sigma[l, r)$ and $\mu[l, r)$ values for each node of $\mathcal{T}$ by first initializing the leaves as follows $\sigma[i, i+1) \leftarrow p_i^-$ and $\mu[i, i+1) \leftarrow p_i^- - r_{i+1}$.

All the $\sigma[l,r)$ and $\mu[l,r)$ values in $\mathcal{T}$ must be updated each time some p_i^- is decreased. They can be maintained in $O(\log n)$ time, as shown by this complete formulation of `decrease`(i,δ), also based on equations 2 and 3 :

procedure `decrease`(i,δ)
begin
  $p_i^- \leftarrow p_i^- - \delta$
  `decrease_rec`$((1,N+1),i,\delta)$
end

procedure `decrease_rec`$((l,r),i,\delta)$
begin
  $\sigma[l,r) \leftarrow \sigma[l,r) - \delta$
  if $l = r-1$ **then**
    if $\sigma[l,r) = 0$ **then** $\mu[l,r) \leftarrow \infty$
        else $\mu[l,r) \leftarrow \mu[l,r) - \delta$
  else
    $m \leftarrow \lceil \frac{l+r}{2} \rceil$
    if $i < m$ **then** `decrease_rec`$((l,m),i,\delta)$
        else `decrease_rec`$((m,r),i,\delta)$
    $\mu[l,r) \leftarrow \min\left(\mu[l,m);\, \sigma[l,m) + \mu[m,r)\right)$
  endif
end

Since $\Delta_{fj^\star} = r_f + \sigma[f,i^\star) + \mu[i^\star,n+1)$, we need two functions able to calculate for any i in the current block the values $\sigma[f,i)$ and $\mu[i,n+1)$. The $\sigma[f,i^\star)$-value does not depend of $j^\star$. $j^\star$ is the operation that minimizes the second term $\mu[i^\star,n+1)$. This value does not depend of f. This decomposition is the base of the query functions.

Since $\sigma[f,i) = \sigma[1,i) - \sigma[1,f)$, we only present a function to compute $\sigma[1,i)$ in $O(\log n)$ time :

function `sigma`$((l,r),i)$
begin
  $m \leftarrow \lceil \frac{l+r}{2} \rceil$
  if $m = r$ **then**
    return $\sigma[l,r)$
  else
    if $i < m$ **then return** `sigma`$((l,m),i)$
        else return $\sigma[l,m) +$ `sigma`$((m,r),i)$
  endif
end

The function to calculate $\mu[i,n+1) =$ `mu`$((1,n+1),i)$ is quite similar. In order to shorten the algorithm description, we do not mention explicitely how $j^\star$ should be returned at the same time as $\mu[i,n+1)$.

function `mu`$((l,r),i)$
begin
  $m \leftarrow \lceil \frac{l+r}{2} \rceil$
  if $m = l$ **then**

```
      return μ[l,r)
   else
      if i < m then return min(mu((l,m),i); σ[l,m) + μ[m,r))
                else  return σ[l,m) + mu((m,r),i)
   endif
end
```

Therefore, we have shown the following lemma.

Lemma 10. *At each call of* `schedule_block_rec`*, the tree data structure* $\mathcal{T}$ *to which are associated the values* $\iota[l,r)$*,* $\sigma[l,r)$ *and* $\mu[l,r)$ *:*

- *is maintained in* $O(\log(n))$ *time ;*
- *finds* $i^\star$ *and* $q_{i^\star}$ *in* $O(\log n)$ *time ;*
- *finds* $j^\star$ *and* $\Delta_{fj^\star}$ *in* $O(\log(n))$ *time.*

Moreover, the data structure can be initialized in linear time.

A.4 Complexity of the algorithm

We finish the proof of Theorem 4 by the analysis of the complexity of `schedule_block`. Each operation is added only once to $\mathcal{A}$ so that the global complexity of all the execution of $\mathcal{A} \leftarrow \mathcal{A} \cup \{k\}$ is $O(n \log n)$. From Lemma 10, it is clear that all the other operations in `schedule_block` are scheduled in $O(\log n)$ time. For any value of $i^\star$ or $j^\star$, each event "$t = T - p_{i^\star}$" and "$t = T - \Delta_{fj^\star}$" can happen at most once: after the event "$t = T - p_{i^\star}$", $i^\star$ is completely scheduled and not available anymore and after the event "$t = T - \Delta_{fj^\star}$", $j^\star$ becomes the first operation of a new current block $\mathcal{B}_{j^\star}$ that is immediately scheduled.

Finally, this proves that $1|prmp, r_i, d_i, q_i|C_{max}$ can be solved in $O(n \log n)$ time.

References

[Bak74] K.R. Baker, *Introduction to sequencing and scheduling*, Wiley & Sons, 1974.

[BLLK83] K.R. Baker, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, *Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints*, Operations Research **26** (1983), 111–120.

[Car82] J. Carlier, *One machine problem*, European Journal of Operational Research **11** (1982), 42–47.

[CP89] J. Carlier and E. Pinson, *An algorithm for solving the job-shop problem*, Management Science **35** (1989), no. 2, 164–176.

[CP98] J. Carlier and E. Pinson, *Jackson's pseudo-preemptive schedule for the* $Pm|r_i, q_i|C_{max}$ *scheduling problem*, Annals of Operations Research **83** (1998), 41–58.

[DL93] J. Du and J.Y.T. Leung, *Minimizing mean flow time with release time and deadline constraints*, Journal of Algorithms **14** (1993), 45–68.

[FG86] A. Federgruen and G. Groenevelt, *Preemptive scheduling of uniform machines by ordinary network flow techniques*, Management Science **32** (1986), 341–349.

[GJ77] M.R. Garey and D.S. Johnson, *Two-processor scheduling with start times and deadlines*, SIAM Journal on Computation **6** (1977), 416–426.

[GLLK79] R.E. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, *Optimization and approximation in deterministic sequencing and scheduling*, Annals of Discrete Mathematics **4** (1979), 287–326.

[GPW97] V. Gordon, E. Potapneva, and F. Werner, *Single machine scheduling with deadlines, release and due dates*, Optimization **42** (1997), 219–244.

[HP94] A.M.A. Hariri and C.N. Potts, *Single machine scheduling with deadlines to minimize the weighted number of tardy jobs*, Management Science **40** (1994), 1712–19.

[Jac55] J.R. Jackson, *Scheduling a production line to minimize manimum tardiness*, Research Report SPIKE-1989-2, Management Science Research Project, University of California, Los Angeles, CA, 1955.

[Law73] E.L. Lawler, *Optimal sequencing of a single machine subject to precedence constraints*, Management Science **19** (1973), 544–546.

[LL78] E.L. Lawler and J. Labetoulle, *On preemptive scheduling on unrelated paralel processors by linear programming*, Journal of the ACM **25** (1978), 612–619.

[NtL98] W. Nuijten and C. Le Pape, *Constraint-based job shop scheduling with* ILOG SCHEDULER, Journal of Heuristics **3** (1998), 271–286.

[Pin95] M. Pinedo, *Scheduling : theory, algorithms, and systems*, Prentice-Hall, 1995.