TO BE PUBLISHED IN OPERATIONS RESEARCH LETTERS

An $O(n^4)$ Algorithm for Preemptive Scheduling of a Single Machine to Minimize the Number of Late Jobs

Philippe Baptiste [1]

UMR CNRS 6599, HEUDIASYC, Université de Technologie de Compiègne

and

Bouygues, Direction des Technologies Nouvelles

Abstract: We study the problem of minimizing, in the preemptive case, the number of late jobs on a single machine ($1 \mid pmtn, r_j \mid \Sigma U_j$). This problem can be solved by Lawler's algorithm [8] whose time and space complexities are respectively $O(n^5)$ and $O(n^3)$. We propose a new dynamic programming algorithm whose complexities are respectively $O(n^4)$ and $O(n^2)$.

Keywords: Single machine scheduling, preemptive scheduling, dynamic programming.

---

[1] Address for correspondence

UMR CNRS 6599, HEUDIASYC,

Université de Technologie de Compiègne, Centre de Recherches de Royallieu,

Rue Personne de Roberval, BP 20.529,

60205 Compiègne Cedex, France

URL: http:\\www.hds.utc.fr\~baptiste, e-mail: baptiste@utc.fr

[2] Acknowledgments.

The author would like to thank Jacques Carlier, Claude Le Pape, Laurent Peridy, Eric Pinson and Marc Sevaux for many enlightening discussions on the topic of this paper. The author gratefully acknowledges the useful comments from the reviewer.

1. Introduction

Given a set of jobs $P = \{J_1, ..., J_n\}$, each job being described by a release date r_i, a due date d_i, and a processing time p_i (with $r_i + p_i \leq d_i$), minimizing the number of late jobs on a single machine consists of finding a schedule of the machine on which the number of jobs scheduled before their due date is maximal ($1 \mid r_j \mid \Sigma U_j$). This problem is NP-hard in the strong sense and is therefore unlikely solvable in polynomial time [6].

However, some special cases are solvable in polynomial time. Moore's well-known algorithm [9] solves in $O(n \log(n))$ steps the special case where release dates are equal. Moreover, when release and due dates of jobs are ordered similarly ($r_i < r_j \Rightarrow d_i \leq d_j$), the problem is solvable in a quadratic amount of steps ([7]).

Lower bounding techniques have also been developed for the general problem. Dauzère-Pérès [5] relies on the resolution of a relaxed Mixed-Integer Linear Program to compute a lower bound of the number of late jobs. Another lower bound can be obtained by relaxing the non-preemption constraint. Lawler [8] has proposed a strongly polynomial algorithm for the preemptive problem ($1 \mid pmtn, r_j \mid \Sigma U_j$). Time and space bounds of this algorithm are respectively $O(n^3k^2)$ and $O(nk^2)$, where k is the number of distinct release dates. So, $O(n^5)$ and $O(n^3)$ if all release dates are distinct. Notice that Lawler's algorithm also applies for minimizing the weighted number of late jobs ($1 \mid pmtn, r_j \mid \Sigma w_j U_j$). It then becomes pseudo-polynomial in the sum W of the weights of the jobs; the bounds being respectively $O(nk^2W^2)$ and $O(k^2W)$.

In this paper, we propose a dynamic programming algorithm for the preemptive and non-weighted version of this problem. It improves the time and space complexities of Lawler's algorithm to respectively $O(n^4)$ and $O(n^2)$. Experimental results show that the optimum can be computed in a small amount of time for relatively large sets of jobs.

This paper is organized as follows. In Section 2, we present a reformulation of the problem and we introduce some notations. Three fundamental propositions are introduced in Section 3. They are the basis of the $O(n^4)$ dynamic programming algorithm described in section 4. We show in Section 5 that this algorithm cannot be extended to solve the weighted version of the problem. Finally, we report some experimental results and we propose some further research directions in Section 6.

2. Reformulation of the Problem

As noticed in [8], the problem reduces to finding a maximum subset of jobs that is feasible, *i.e.*, which can be preemptively scheduled on a single machine. It is well known that testing the feasibility of a subset O of jobs can be achieved by computing JPS_O, the Jackson Preemptive Schedule of O. JPS_O is the preemptive schedule obtained by applying the Earliest Due Date priority dispatching rule: whenever the machine is free and one job in O is available, schedule the job $J_i \in O$ for which d_i is the smallest. If a job J_j becomes available while J_i is in process, stop J_i and start J_j if d_j is strictly smaller than d_i, otherwise continue J_i. The following properties of Jackson Preemptive Schedule are of great interest (for a proof, see for instance [2]).

- If a job is scheduled on JPS_O after its due date, O is not feasible.
- The makespan C_O (*i.e.*, the time at which all activities are finished) of JPS_O is minimal among all preemptive schedules.

From now on, we assume that jobs are sorted in increasing order of their due dates ($d_1 \leq d_2 \leq \ldots \leq d_n$). For any integer a and any job J_k, let $S_k(a)$ be the set of jobs J_i such that $a \leq r_i$ and $i \leq k$. Given an interval $[a, b]$ and a set of jobs O, $slack(O, a, b)$ denotes the time during which JPS_O is idle over $[a, b]$. By convention, $C_\varnothing = -\infty$.

Let a and b be any values such that $a \leq b$. We introduce three definitions. As we will see later on, we will be mainly interested in the values of a and b that are release dates.

Let $C_k(a, m)$ be the minimal time at which m jobs in $S_k(a)$ can be completed.

$$C_k(a, m) = \min\{\{\infty\} \cup \{C_O \mid O \subseteq S_k(a), O \text{ feasible and } |O| = m\}\}$$

Let $\pi_k(a, b)$ be the maximal number of jobs in $S_{k-1}(a)$ that can be scheduled before b.

$$\pi_k(a, b) = \max\{|O| \mid O \subseteq S_{k-1}(a), O \text{ feasible and } C_O \leq b\}$$

For any $b \geq r_k$, let $\mu_k(a, b)$ be the largest possible slack over the interval $[r_k, b]$ among sets that realize $\pi_k(a, b)$.

$$\mu_k(a, b) = \max\{slack(O, r_k, b) \mid O \subseteq S_{k-1}(a), O \text{ feasible}, |O| = \pi_k(a, b), C_O \leq b\}$$

3. Some Fundamental Properties

We prove three propositions that exhibit a strong link between the values of C, π and μ. We first show that C_k can be computed in function of C_{k-1}, π_k and μ_k. To write a compact formula, we introduce $f_k(x)$ which is equal to x if $x \leq d_k$ and to $+\infty$ otherwise.

Proposition 1. If $J_k \notin S_k(a)$ (*i.e.*, $r_k < a$) then $C_k(a, m) = C_{k-1}(a, m)$. If $J_k \in S_k(a)$ then

$$C_k(a, m) = f_k(\min(C_{k-1}(a, m),$$
$$\max(r_k, C_{k-1}(a, m - 1)) + p_k,$$
$$\min_{r_u \geq r_k} (C_{k-1}(r_u, m - 1 - \pi_k(a, r_u)) + \max(0, p_k - \mu_k(a, r_u)))))$$

Proof. The first part of the proposition is obvious. Consider now that $J_k \in S_k(a)$. Let C' be the value corresponding to the right term in the equation above.

We first prove that $C' \leq C_k(a, m)$. We can suppose that $C_k(a, m)$ has a finite value (if not, the result is obvious). Several cases can occur:

- Either there is a set O that realizes $C_k(a, m)$ such that $J_k \notin O$. Then $C_k(a, m)$ is equal to $C_{k-1}(a, m)$. Consequently, $C' \leq C_k(a, m)$.
- Or J_k belongs to all the sets that realize $C_k(a, m)$ but there is one, say O, such that J_k is fully executed on JPS_O after all other jobs. Then, $C_k(a, m) = \max(C_{O - \{Jk\}}, r_k) + p_k$. Moreover, $C_{O - \{Jk\}} \geq C_{k-1}(a, m - 1)$. Thus, $C_k(a,m) \geq \max(r_k, C_{k-1}(a,m-1)) + p_k \geq C'$.
- Or J_k belongs to all the sets that realize $C_k(a, m)$ and J_k is never fully executed after all the other jobs. Let O be a set that realizes $C_k(a, m)$ and let t be the maximal time point such that (a) J_k executes in $[t - 1, t]$ on JPS_O and (b) another job executes after time t on JPS_O. Given our hypothesis, such a time point exists. Moreover, because of the particular structure of Jackson Preemptive Schedules, the first time point at which an activity executes after t is the release date r_u of a job J_u (with $r_u \geq r_k$).

  Consider now a job J_i ($i \neq k$) which starts on JPS_O before r_u. Since $d_i < d_k$, J_i ends before r_u (otherwise J_i would be scheduled at time $t - 1$ on JPS_O instead of J_k). Let then O_1 be the set of jobs in $O - \{J_k\}$ that end before r_u on JPS_O. Let O_2 be the set of jobs in $O - \{J_k\}$ that start after or at r_u.

  If O_1 is not maximal (*i.e.*, $|O_1| < \pi_k(a, r_u)$), then consider the set O'_1 that realizes $\pi_k(a,r_u)$. It is obvious that $O'_1 \cup O_2$ is feasible and that it contains as many jobs as O. Moreover, $C_{O'1 \cup O2}$ is lower than or equal to C_O; which contradicts our hypothesis that J_k is in all the sets that realize $C_k(a, m)$. We know that O_1 contains $\pi_k(a, r_u)$ jobs, moreover $slack(O_1, r_k, r_u)$ time units are available before r_u to schedule J_k. Thus, $C_O = \max(0, p_k - slack(O_1, r_k, r_u)) + C_{O2} \geq C_{k-1}(r_u, m - 1 - \pi_k(a,r_u)) + \max(0, p_k - \mu_k(a,r_u)) \geq C'$

We now prove that $C' \geq C_k(a, m)$. Notice that if $C' = \infty$, the result is obvious. We can then suppose that C' is the minimum of one of the three terms.

- If $C' = C_{k-1}(a, m)$. Since $C_k(a, m) \leq C_{k-1}(a, m)$, $C' \geq C_k(a, m)$.

- If $C' = \max(r_k, C_{k-1}(a, m\text{-}1)) + p_k$. Let O be the set that realizes $C_{k-1}(a, m\text{-}1)$. J_k is not late if it is "added" at the end of JPS_O (because C' is finite). Thus, the set $O \cup \{J_k\}$ contains m jobs and is feasible. Moreover, it is a subset of $S_k(a)$. As a consequence, $C' \geq C_{O \cup \{Jk\}} \geq C_k(a, m)$.
- If $\exists\, r_u \geq r_k \mid C' = C_{k-1}(r_u, m - 1 - \pi_k(a, r_u)) + \max(0, p_k - \mu_k(a, r_u))$. Let O_1 be the set that realizes $\mu_k(a, r_u)$ and O_2 be the set that realizes $C_{k-1}(r_u, m - 1 - \pi_k(a, r_u))$. First, notice that $O_1 \cup O_2 \cup \{J_k\}$ obviously belongs to $S_k(a)$. Second, notice that $O_1 \cup O_2 \cup \{J_k\}$ is feasible. Indeed, on $JPS_{O1 \cup O2}$, there are $\mu_k(a, r_u)$ "holes" between r_k and r_u and thus, the quantity $C_{k-1}(r_u, m - 1 - \pi_k(a, r_u)) + \max(0, p_k - \mu_k(a, r_u))$ is the makespan of $JPS_{O1 \cup O2}$ on which J_k has been scheduled as soon as possible in the "holes". This makespan is lower than d_k (otherwise $C' = \infty$). Third, notice that there are m jobs in $O_1 \cup O_2 \cup \{J_k\}$. Indeed, according to the definition of μ, before r_u, $\pi_k(a, r_u)$ jobs are scheduled. After r_u, $m - 1 - \pi_k(a, r_u)$ jobs are scheduled; which means that $m - 1$ jobs in $O_1 \cup O_2$ are scheduled.

We have proven that $C' \geq C_{O1 \cup O2 \cup \{Jk\}}$; thus $C' \geq C_k(a, m)$. □

Proposition 2. For any $a \leq b$, $\pi_k(a, b) = \max\{m \mid C_{k-1}(a, m) \leq b\}$.

Proof. Let $\pi' = \max\{m \mid C_{k-1}(a, m) \leq b\}$.

Let O be the set that realizes $C_{k-1}(a, \pi')$. O is a subset of $S_{k-1}(a)$, O is feasible and $C_O = C_{k-1}(a, \pi') \leq b$; consequently, $\pi' \leq \pi_k(a, b)$.

Let O be the set that realizes $\pi_k(a, b)$. According to the definition of C, $C_k(a, |O|) \leq C_O$. Consequently $C_k(a, |O|) \leq b$ and thus, $\pi' \geq \pi_k(a, b)$. □

Proposition 3. $\forall\ a, b$ and $\forall\ J_k$ such that $a \leq r_k \leq b$,

$$\mu_k(a, b) = \max(b - \max(C_{k-1}(a, \pi_k(a, b)), r_k),$$

$$\max_{\begin{cases} r_k \leq r_v < b \\ \pi_k(a,b) = \pi_k(a,r_v) + \pi_k(r_v,b) \\ \pi_k(r_v,b) > 0 \end{cases}} (\mu_k(a, r_v) + b - C_{k-1}(r_v, \pi_k(r_v, b))))$$

Proof. Let μ' be the value corresponding to the right term in the previous equation.

We first prove that $\mu' \geq \mu_k(a, b)$. Let O be the set that realizes $\mu_k(a, b)$. The proof relies on the fact that if there is a time point $t \in [r_k, b]$ such that JPS_O is idle in $[t - 1, t]$, the computation of the maximal slack can be decomposed into the computation of the maximal slack over $[r_k, t]$ and over $[t, b]$. In the following, we distinguish two cases.

- If $C_O \leq r_k$, then $slack(O, r_k, b) = b - r_k$. Moreover, $C_{k-1}(a, \pi_k(a, b)) \leq C_O \leq r_k$. Thus, $b - \max(C_{k-1}(a, \pi_k(a, b)), r_k)$ is equal to $b - r_k$. Consequently, $\mu' \geq \mu_k(a, b)$.
- If $C_O > r_k$, let t be the largest time point such that JPS_O is idle immediately before t and never idle in the interval $[t, C_O]$. According to the structure of a Jackson Preemptive Schedule, t is a release date; say r_v. Two cases are distinguished.

  First, if $r_v \leq r_k$ then $slack(O, r_k, b) = b - C_O \leq b - C_{k-1}(a, \pi_k(a, b))$; thus $\mu' \geq \mu_k(a, b)$. Second, suppose that $r_v > r_k$. According to the definition of r_v, $\pi_k(r_v, b) > 0$. We claim that $\pi_k(a, b) = \pi_k(a, r_v) + \pi_k(r_v, b)$. Indeed, consider O_1 the subset of O that consists of the jobs ending before or at r_v on JPS_O and O_2 the subset of O that consists of the jobs ending after r_v on JPS_O. On the one hand, $|O_1| + |O_2| = |O| = \pi_k(a, b)$, on the other hand, $|O_1| \leq \pi_k(a, r_v)$ and $|O_2| \leq \pi_k(r_v, b)$. Suppose that the first inequality is strict, let then O'_1 be the set that realizes $\pi_k(a, r_v)$. It is easy to see that the set $O'_1 \cup O_2$ is included in $S_{k-1}(a)$, is feasible and that its Jackson Preemptive Schedule ends before b. Moreover, $O'_1 \cup O_2$ is larger than O, which contradicts the fact that O realizes $\pi_k(a, b)$; consequently, $|O_1| = \pi_k(a, r_v)$. Similarly, we can prove that $|O_2| = \pi_k(r_v, b)$.

Let us compute the slack of O over $[r_k, b]$. $slack(O, r_k, b) = slack(O_1, r_k, r_v) + slack(O_2, r_v, b)$. As a consequence, $slack(O, r_k, b) \leq \mu_k(a, r_v) + b - C_{k-1}(r_v, \pi_k(r_v, b)) \leq \mu'$.

We now prove that $\mu' \leq \mu_k(a, b)$. Let us distinguish the two following cases:

- If $\mu' = b - \max(r_k, C_{k-1}(a, \pi_k(a, b)))$, let then O be the set that realizes $C_{k-1}(a, \pi_k(a, b))$. JPS_O is idle after $C_{k-1}(a, \pi_k(a, b))$ and its makespan is lower than or equal to b, thus $slack(O, r_k, b) \geq b - \max(C_{k-1}(a, \pi_k(a, b)), r_k)$. Moreover, $O \subseteq S_{k-1}(a)$, O is feasible, $|O| = \pi_k(a, b)$ and $C_O \leq b$; thus $\mu_k(a, b) \geq slack(O, r_k, b)$. Consequently, $\mu' \leq \mu_k(a, b)$.
- If there is a release date r_v such that (1) $r_k \leq r_v < b$, (2) $\pi_k(a, b) = \pi_k(a, r_v) + \pi_k(r_v, b)$, (3) $\pi_k(r_v, b) > 0$ and (4) $\mu' = \mu_k(a, r_v) + b - C_{k-1}(r_v, \pi_k(r_v, b))$, let then O_1 be the set that realizes $\mu_k(a, r_v)$ and let O_2 be the set that realizes $C_{k-1}(r_v, \pi_k(r_v, b))$. Notice that $slack(O_1 \cup O_2, r_k, b) \geq \mu'$ since the quantity $\mu_k(a, r_v) + b - C_{k-1}(r_v, \pi_k(r_v, b))$ is the slack of O_1 before r_v plus a lower bound of the slack of O_2 in $[r_v, b]$ ($\pi_k(r_v, b) > 0$ ensures that $C_{k-1}(r_v, \pi_k(r_v, b))$ is finite). Moreover, O_1 and O_2 are disjoint because $\forall J_i \in O_1, r_i < r_v$ and $\forall J_i \in O_2, r_i \geq r_v$. Thus, $|O_1 \cup O_2| = \pi_k(a, r_v) + \pi_k(r_v, b) = \pi_k(a, b)$. It is easy to verify that $O_1 \cup O_2 \subseteq S_{k-1}(a)$, that $O_1 \cup O_2$ is feasible and that $C_{O1 \cup O2} \leq b$; thus $\mu_k(a, b) \geq slack(O_1 \cup O_2, r_k, b)$. Consequently, $\mu' \leq \mu_k(a, b)$. □

Propositions 1, 2, 3 are the basis of the dynamic programming algorithm that we propose in the following section.

4. Overall Algorithm

Our aim is to determine the largest value of m such that $C_n(\min_i r_i, m)$ is finite. The variables of the dynamic programming algorithm correspond to $C_k(a, m)$, $\pi_k(a, b)$ and $\mu_k(a, b)$. They are stored in multi-dimensional arrays. Actually, it is easy to understand that given Propositions 1 and 3, the relevant values of a and b are those corresponding

to release dates. Thus, the values of $C_k(a, m)$, $\pi_k(a, b)$ and $\mu_k(a, b)$ are stored in indexed arrays (*e.g.*, $C_k(r_j, m)$ is stored in a 3-dimensional array at the "position" (k, j, m)).

The first step of the algorithm is the computation of $C_1(r_j, m)$ for all release date r_j and all the values of m in $[1, n]$.

- If $m = 0$, $C_1(r_j, 0) = -\infty$
- If $m = 1$ and $r_1 < r_j$, $C_1(r_j, 1) = \infty$
- If $m = 1$ and $r_1 \geq r_j$, $C_1(r_j, 1) = r_1 + p_1$
- If $m > 1$, $C_1(r_j, m) = \infty$

The second step is a loop on k from 2 to n.

- For each release date r_j and each release date r_u, compute $\pi_k(r_j, r_u)$. This computation is done in $O(n)$ thanks to Proposition 2.
- For each release date r_j and each release date r_u (taken in increasing order), compute the values of $\mu_k(r_j, r_u)$. This is done in $O(n)$ thanks to Proposition 3. Indeed, for a given value of r_u, we use the pre-computed values of $\mu_k(r_j, r_v)$ (with $r_v < r_u$). Moreover, the tests $r_k \leq r_v < r_u$, $\pi_k(r_j, r_u) = \pi_k(r_j, r_v) + \pi_k(r_v, r_u)$ and $\pi_k(r_v, r_u) > 0$ are computed in constant time.
- For each release date r_j and each value of m, compute $C_k(r_j, m)$. This is done in $O(n)$ thanks to Proposition 1.

The overall algorithm then runs in $O(n^4)$. A rough analysis in terms of memory consumption leads to an $O(n^3)$ bound. Indeed, three cubic arrays are needed to store the values of $C_k(r_j, m)$, $\pi_k(r_j, r_u)$ and $\mu_k(r_j, r_u)$. However, notice that at each step of the outer loop on k, one only needs the values of C computed at the previous step (k-1). Thus, the algorithm can be implemented with 4 arrays of $n*n$ size (one for π, one for μ, one for the previous values of C and one for the current values of C); which leads to a space complexity of $O(n^2)$.

This algorithm does not exhibit a set O that realizes the optimum of the problem. A backward computation can be done to determine such a set. The space complexity then increases to $O(n^3)$ since all values taken by C must be stored. Since we are mainly interested in the computation of the optimum, which serves as a lower bound of the non-preemptive problem, we do not provide the description of how O can be computed.

5. Minimizing the Weighted Number of Late Jobs...

At this point, an interesting question is whether our algorithm can be extended to solve the weighted version of the problem (*i.e.*, a version of the problem where each job J_i has a weight $w_i \geq 0$ and where the goal is to minimize the weighted number of late jobs). The definitions of the variables C, π and μ can be easily extended:

- $C_k(a, w)$ is the minimal time at which a set of jobs in $S_k(a)$, whose weight is greater than or equal to w, can be completed (if no such set exists, $C_k(a, w) = \infty$).
- $\pi_k(a, b)$ is the maximal weighted number of late jobs in $S_{k-1}(a)$ that can be scheduled before b.
- $\mu_k(a, b)$ is the largest possible slack over $[r_k, b]$ among sets that realize $\pi_k(a, b)$.

Given these definitions, one could think that Proposition 1 can be extended as follows:

If $J_k \notin S_k(a)$ (*i.e.*, $r_k < a$) then $C_k(a, w) = C_{k-1}(a, w)$. If $J_k \in S_k(a)$ then

$$C_k(a, w) = f_k(\min(C_{k-1}(a, w),$$
$$\max(r_k, C_{k-1}(a, w - w_k)) + p_k,$$
$$\min_{r_u \geq r_k} (C_{k-1}(r_u, w - w_k - \pi_k(a, r_u)) + \max(0, p_k - \mu_k(a, r_u)))))$$

Unfortunately, this extension does not hold. Intuitively, this comes from the fact that, in Proposition 1, the expression $\min_{r_u \geq r_k} (C_{k-1}(r_u, m - 1 - \pi_k(a, r_u)) + \max(0, p_k - \mu_k(a, r_u)))$ means that that it is worth scheduling between a and r_u a maximum number of jobs in $S_{k-1}(a)$. On the contrary, if jobs are weighted, it can be of interest to schedule a smaller amount of jobs (in term of weights) between a and r_u to increase the slack and thus to leave more space to schedule J_k.

The following counter-example illustrates this phenomenon. Consider five weighted jobs $J_1(r_1=0,p_1=3,d_1=6,w_1=2)$, $J_2(r_2=0,p_2=3,d_2=6,w_2=2)$, $J_3(r_3=0,p_3=2,d_3=6,w_3=1)$, $J_4(r_4=6,p_4=1,d_4=7,w_4=2)$, and $J_5(r_5=0,p_5=3,d_5=9,w_5=10)$. It is easy to prove by hand that $C_5(0, 15) = 9$. This is not the result obtained when applying the weighted version of Proposition 1:

$$C_5(0, 15) = f_5(\min(C_4(0, 15),$$
$$\max(0, C_4(0, 15 - 10)) + 3,$$
$$C_4(6, 15 - 10 - \pi_4(0, 6)) + \max(0, 3 - \mu_4(0, 6)))$$
$$= f_5(\min(\infty, \max(0, 7) + 3, C_4(6, 1) + \max(0, 3)))$$
$$= f_5(\min(\infty, 10, 7 + 3)) = \infty$$

6. Conclusion and Future Work

Our algorithm has been implemented in CLAIRE [4] and tested on a set of 50 instances generated by Marc Sevaux, each of them containing 50 jobs. On average, an instance is solved in 0.6 seconds of CPU time on a PC Dell OptiPlex GX Pro running Windows NT. The algorithm has also been tested on larger instances containing 100 jobs each. These instances are obtained by duplicating each job of the instances of the first set. The average CPU time to solve the new instances is 7.9 seconds. These experimental results show that, in practice, relatively large instances can be solved in a reasonable amount of CPU time. Several questions are open at this point.

First, it would be interesting to compare, both from a theoretical and from an experimental point of view, the quality of the preemptive lower bound to other lower bounds, either obtained by the resolution of a linear program (*e.g.*, [5]) or obtained by another relaxation of the problem (*e.g.*, a relaxation of the release dates [7]).

Second, we think that the preemptive relaxation can play a central part in the resolution of the non-preemptive problem, as it plays a central part in the resolution of the "classical" one machine problem known as $(1 \mid r_j, q_j \mid C_{max})$ (*e.g.*, [1], [2], [3]). For this problem, the preemptive relaxation is not only used to compute a lower bound of the makespan but also to determine necessary conditions that a schedule must satisfy if it is completed before a given deadline D. We think that for the $(1 \mid r_j \mid \Sigma\ U_j)$ problem, similar necessary conditions can be derived. Given an upper bound ub of the optimum, we could try to determine

a) the jobs that must be on time on any preemptive schedule with less than ub late jobs,
b) the jobs that must be late on any preemptive schedule with less than ub late jobs,
c) the earliest (respectively latest) time point x_i at which a given job J_i can end (respectively start) on a schedule with less than ub late jobs.

Notice that Potts and Van Wassenhove ([10]) rely on necessary conditions closely related to a) and b) for solving in an efficient way the weighted and non-preemptive version of the problem with identical release dates ($1 \mid \mid \Sigma\ w_j\ U_j$). The necessary condition c) is of great interest since the computation of x_i would result in the adjustment of the release date r_i of the job J_i to x_i - p_i. Such a mechanism would mimic the time-bound adjustment schemes proposed for the one-machine problem ([2], [3]). Of course, a lot of work remains to build efficient algorithms that implement such necessary conditions.

References

[1] P. Brucker, B. Jurisch and B. Sievers, "A Branch and Bound Algorithm for the Job-Shop Scheduling Problem", *Discrete Applied Mathematics*, **49(1)**, 107-127 (1994).

[2] J. Carlier, "The One-Machine Sequencing Problem", *European Journal of Operational Research* **11**, 42-47 (1982).

[3] J. Carlier and E. Pinson, "An Algorithm for Solving the Job-Shop Problem", *Management Science* **35(2)**, 164-176 (1989).

[4] Y. Caseau and F. Laburthe, "CLAIRE: A Parametric Tool to Generate C++ Code for Problem Solving", *Working Paper, Bouygues, Direction Scientifique* (1996).

[5] S. Dauzère-Pérès, "Minimizing Late Jobs in the General One Machine Scheduling Problem", *European Journal of Operational Research* **81**, 134-142 (1995).

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, (1979).

[7] H. Kise, T. Ibaraki and H. Mine, "A Solvable Case of the One-Machine Scheduling Problem with Ready and Due Times", *Operations Research* **26(1)**, 121-126 (1978).

[8] E. L. Lawler, "A Dynamic Programming Algorithm for Preemptive Scheduling of a Single Machine to Minimize the Number of Late Jobs", *Annals of Operations Research* **26**, 125-133 (1990).

[9] J.M. Moore, "An *n* job, one machine sequencing algorithm for minimizing the number of late jobs", *Management Science* **15(1)**, 102-109 (1968).

[10] C.N. Potts and L.N. Van Wassenhove, "Algorithms for scheduling a single machine to minimize the weighted number of late jobs", *Management Science* **34(7)**, 843-858 (1988).