TO APPEAR IN ANNALS OF OPERATIONS RESEARCH

Satisfiability Tests and Time-Bound Adjustments for Cumulative Scheduling Problems

PHILIPPE BAPTISTE[1,2], CLAUDE LE PAPE[1] AND WIM NUIJTEN[3]

[1]*Bouygues, Direction des Technologies Nouvelles,*
1, av. E. Freyssinet, Saint-Quentin-en-Yvelines, F-78061
[2]*UMR CNRS 6599 HEUDIASYC, Université de Technologie de Compiègne*
[3]*Ilog SA, 9, rue de Verdun, Gentilly, F-94253*

Philippe.Baptiste@hds.utc.fr, **lepape@dmi.ens.fr**, **nuijten@ilog.fr**

This paper presents a set of satisfiability tests and time-bound adjustment algorithms that can be applied to cumulative scheduling problems. An instance of the Cumulative Scheduling Problem (CuSP) consists of (1) one resource with a given capacity and (2) a set of activities, each having a release date, a deadline, a processing time and a resource capacity requirement. The problem is to decide whether there exists a start time assignment to all activities such that at no point in time the capacity of the resource is exceeded and all timing constraints are satisfied. The Cumulative Scheduling Problem can be seen as a relaxation of the decision variant of the Resource-Constrained Project Scheduling Problem.
We present three necessary conditions for the existence of a feasible schedule. Two of them are obtained by polynomial relaxations of the CuSP. The third one is based on energetic reasoning. We show that the second condition is closely related to the subset bound, a well-known lower bound of the m-Machine Problem. We also present three algorithms, based on the previously mentioned necessary conditions, to adjust release dates and deadlines of activities. These algorithms extend the time-bound adjustment techniques developed for the One-Machine Problem. They have been incorporated in a branch and bound procedure to solve the Resource-Constrained Project Scheduling Problem. Computational results are reported.

Keywords: cumulative scheduling, resource constraints, resource-constrained project scheduling, time-bound adjustments.

1 Introduction and Notations

Roughly speaking, scheduling is the process of assigning activities to resources in time. Two classes of scheduling problems can be distinguished. In the **disjunctive** scheduling class, each resource can execute at most one activity at a time. For example, the One-Machine Problem, the Flow-Shop Problem and the Job-Shop Problem belong to the disjunctive scheduling class. In the **cumulative** scheduling class, a resource can execute several activities in parallel, provided that the resource capacity is not exceeded. The Multiprocessor Flow-Shop Problem, the Multiprocessor Job-Shop Problem and the Resource-Constrained Project Scheduling Problem belong to the cumulative scheduling class. Most of the successful exact approaches to minimize makespan within the disjunctive scheduling class rely on an

extensive use of satisfiability tests and of time-bound adjustments for the One-Machine Problem ([2, 3, 6, 9, 12, 24, 25, 29]). The satisfiability tests often consist in verifying that there exists a preemptive schedule. The time-bound adjustments allow to tighten the release dates and the deadlines of the activities. The aim of this paper is to present counterparts of such techniques for the cumulative class.

We consider the Resource-Constrained Project Scheduling Problem (RCPSP), a very general problem in the cumulative scheduling class. An instance of the decision variant of the RCPSP consists of (1) a set of resources of given capacities, (2) a set of non-interruptible activities of given durations, (3) an acyclic network of precedence constraints between the activities, (4) for each activity and each resource the amount of the resource required by the activity over its execution and (5) an overall deadline D. The problem is to find a start time assignment that satisfies the precedence and resource capacity constraints, and whose makespan (*i.e.*, the time at which all activities are completed) is at most D. As a generalization of the Job-Shop Scheduling Problem, the decision variant of the RCPSP is NP-complete in the strong sense [19].

Given an instance of the RCPSP, release dates and deadlines of activities can be derived from the network of precedence constraints and from the overall deadline D, for example using Ford's algorithm [20]. In this paper, we focus on a sub-problem of the RCPSP, in which precedence constraints are relaxed and a single resource is considered at a time. Notice that this sub-problem of the RCPSP is the counterpart of the decision variant of the One-Machine Problem in the disjunctive class. We call this problem the Cumulative Scheduling Problem (CuSP). More formally, an instance of the CuSP consists of (1) one resource with a given capacity C and (2) a set of n activities $\{A_1, ..., A_n\}$, together with a release date r_i, a deadline d_i, a processing time p_i, and a resource capacity requirement c_i for each activity A_i. We assume that all data are integers and that $\forall i$, $r_i + p_i \leq d_i$ and $c_i \leq C$. The problem is to decide whether there exists a feasible schedule, *i.e.*, a start time assignment that satisfies all timing constraints and the resource constraint. The CuSP obviously belongs to NP. It is an extension of the decision variant of both the One-Machine Problem ($C = 1$, $c_i = 1$) and the m-Machine Problem ($C = m$, $c_i = 1$) and thus is NP-complete in the strong sense [19].

As mentioned before, a large amount of work has been carried out on the One-Machine Problem. Similarly, lower bounds have been developed for the optimization variant of the m-Machine Problem (*e.g.*, [11, 28]). Obviously, these lower bounds can be seen as necessary conditions of existence for the decision variant of the m-Machine Problem. As far as we know, no specific algorithm for adjusting release dates and deadlines has been proposed. On the CuSP itself, little work has been done. Constraint propagation algorithms have been developed to adjust time-bounds of activities (*e.g.*, [13, 23, 25, 26]), but they tend to be less uniformly effective than the algorithms available for the One-Machine Problem.

In this paper, we study three necessary conditions of existence for the CuSP.

- The first necessary condition is based on the resolution of the Fully Elastic CuSP, a relaxation of the CuSP. An instance of the Fully Elastic CuSP is described by the same data as an instance of the CuSP. The problem is to decide whether there exists a feasible fully elastic schedule, *i.e.*, an integer function fes(t, i) representing the number of units of the resource assigned to A_i over the interval [t t+1), such that:

$$\forall i, \forall t \notin [r_i\ d_i), \text{fes}(t, i) = 0$$
$$\forall i, \sum_t \text{fes}(t, i) = p_i * c_i$$
$$\forall t, \sum_i \text{fes}(t, i) \leq C$$

- The second necessary condition is based on the resolution of the Partially Elastic CuSP, a tighter relaxation of the CuSP (*cf.* Figure 1). An instance of the Partially Elastic CuSP is described by the same data as an instance of the CuSP. The problem is to decide whether there exists a feasible partially elastic schedule, *i.e.*, an integer function pes(t, i) such that:

$$\forall\, i, \forall\, t \notin [r_i\ d_i),\ pes(t, i) = 0$$
$$\forall\, i, \sum_t pes(t, i) = p_i * c_i$$
$$\forall\, t, \sum_i pes(t, i) \leq C$$
$$\forall\, i, \forall\, t \in [r_i\ d_i), \Sigma_{x<t}\ pes(x, i) \leq c_i * (t - r_i)$$
$$\forall\, i, \forall\, t \in [r_i\ d_i), \Sigma_{t\leq x}\ pes(x, i) \leq c_i * (d_i - t)$$

- The third necessary condition, called the "left-shift/right-shift" necessary condition, is not based on a well-identified relaxation of the CuSP but on energetic reasoning as defined in [17, 23].

For each of these necessary conditions, we propose a polynomial algorithm, running in $O(n * \log(n))$ for the fully elastic condition, and in $O(n^2)$ for the partially elastic and the left-shift/right-shift conditions. In the particular case of the m-Machine Problem, these necessary conditions can be theoretically compared with other results from the literature. The subset bound [28] (seen as a necessary condition) is equivalent to the partially elastic relaxation and the left-shift/right-shift necessary condition is strictly stronger than the partially elastic relaxation.

We also propose three time-bound adjustment schemes for the CuSP.

- The first one is based on the fully elastic relaxation. An $O(n^2)$ algorithm is described.
- The second one is based on the partially elastic relaxation. An $O(n^2 * |\{c_i\}|)$ algorithm is described (where $|\{c_i\}|$ is the number of distinct resource capacity requirements).
- The third one is based on the left-shift/right-shift necessary condition. An $O(n^3)$ algorithm is described.

This paper is organized as follows: Section 2 presents the three necessary conditions; Section 3 describes the three corresponding time-bound adjustment algorithms; Section 4 presents an experimental comparison of these algorithms, made on the basis of a branch and bound procedure for the Resource-Constrained Project Scheduling Problem.

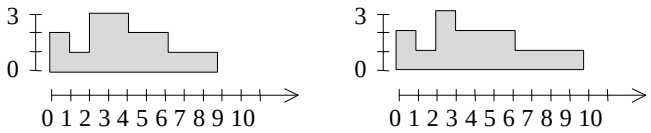


Figure 1: Consider a resource of capacity 3 and an activity with release date 0, deadline 10, processing time 8 and resource requirement 2. Both Gantt charts correspond to feasible fully elastic schedules. Only the right Gantt chart is a feasible partially elastic schedule. Indeed, in the left Gantt chart, 9 units of the resource are used in [0 4), which is more than 2∗(4-0).

2. Necessary Conditions for the Existence of a Feasible Schedule

2.1 A Necessary and Sufficient Condition for the Fully Elastic CuSP

We exhibit in this section a strong link between the Fully Elastic CuSP and the decision variant of the Preemptive One-Machine Problem. An instance of the Preemptive One-

Machine Problem is defined by a set of n activities $\{A_1, ..., A_n\}$, together with a release date r_i, a deadline d_i and a processing time p_i for each activity A_i. The problem is to decide whether there exists a feasible preemptive one-machine schedule of the given activities.

Transformation F.
For any instance I of the Fully Elastic CuSP, let F(I) be the instance of the Preemptive One-Machine Problem defined by n activities $A_1', ..., A_n'$ with $\forall i, r_i' = C * r_i, d_i' = C * d_i, p_i' = p_i * c_i$.

Proposition 1.
For any instance I of the Fully Elastic CuSP, there exists a feasible fully elastic schedule of I if and only if there exists a feasible preemptive schedule of F(I).

Proof.
Let C be the capacity of the resource R of the instance I. Let R' be the resource of the instance F(I). We first prove that if there is a feasible fully elastic schedule of I, then there is a feasible preemptive schedule of F(I). Let fes(t, i) be the number of units of A_i executed at t. We build a schedule of $A_1', ..., A_n'$ on R' as follows. For each time t and each activity A_i, schedule fes(t, i) units of A_i' on R' as early as possible after time $C * t$. It is obvious that at any time t, for any activity A_i, the number of units of A_i executed at t on R is equal to the number of units of A_i' executed between $C * t$ and $C * (t + 1)$ on R' since this algorithm consists of cutting the schedule of $A_1, ..., A_n$ into slices of one time unit and rescheduling these slices on R'. Consequently, for any activity A_i', exactly $p_i * c_i$ units of A_i' are scheduled between $C * r_i$ and $C * d_i$ and thus the release dates as well as deadlines are met. A symmetric demonstration would prove that if there is a feasible preemptive schedule of F(I) then there is a feasible fully elastic schedule of I. □

Consider now Jackson's Preemptive Schedule (JPS), the One-Machine preemptive schedule obtained by applying the Earliest Due Date priority dispatching rule. JPS is feasible if and only if there exists a feasible preemptive schedule. Moreover, JPS can be build in $O(n * \log(n))$ steps (see [8] for details). Consequently, thanks to Proposition 1, we have an $O(n * \log(n))$ algorithm to solve the Fully Elastic CuSP. In the following, Jackson's Fully Elastic Schedule (JFES) denotes the fully elastic schedule obtained (1) by applying JPS on the transformed instance and (2) by rescheduling slices as described in the proof of Proposition 1.

2.2 A Necessary and Sufficient Condition for the Partially Elastic CuSP

The Partially Elastic CuSP is slightly more complex. We first introduce a pseudo-polynomial algorithm to solve this problem. We then present the concept of required energy consumption, which enables us to show that the Partially Elastic CuSP is equivalent to another problem for which we can provide a quadratic algorithm. In the following, "I" denotes an instance of the Partially Elastic CuSP. Let us first introduce a new transformation.

Transformation G.
Consider the instance G(I) of the Fully Elastic CuSP defined by replacing every activity A_i by p_i activities $A_i^1, \ldots, A_i^{p_i}$, each having a resource requirement $c_i^j = c_i$, a release date $r_i^j = r_i + j - 1$, a deadline $d_i^j = d_i - (p_i - j)$ and a processing time p_i^j of 1 (the resource capacity of G(I) is C as for I).

2.2.1 Jackson's Partially Elastic Schedule

Jackson's Partially Elastic Schedule (JPES) is the schedule built by scheduling each activity A_i at the time points at which the activities A_i^j are scheduled on JFES of G(I). Given the definition of G, it is easy to verify that, if JFES is a feasible fully elastic schedule of G(I) then JPES is a feasible partially elastic schedule of I.

Proposition 2.
There exists a feasible partially elastic schedule if and only if JPES is a feasible partially elastic schedule.

Proof (sketch).
Consider a feasible partially elastic schedule S of an instance I. It is then possible to build a feasible fully elastic schedule of G(I) obtained from S by a similar transformation as G (*i.e.*, for any activity A_i, schedule A_i^1 at the same place as the "first c_i units" of A_i on S, iterate ...). Since there is a feasible fully elastic schedule of G(I), JFES is also a feasible fully elastic schedule of G(I) (Proposition 1). Thus, JPES is a feasible partially elastic schedule of I. □
Since transformation G is done in $O(\Sigma p_i)$ and since the fully elastic problem G(I) can be solved in $O(\Sigma p_i * \log(\Sigma p_i))$, Proposition 2 leads to an $O(\Sigma p_i * \log(\Sigma p_i))$ algorithm to test the existence of a feasible partially elastic schedule.

2.2.2 Energetic Reasoning

We adapt the notion of "required energy consumption" defined in [23] to partially elastic activities. The required energy consumption $W_{PE}(A_i, t_1, t_2)$ of an activity over an interval $[t_1\ t_2]$ is defined as follows (*cf.* Figure 2).

$$W_{PE}(A_i, t_1, t_2) = c_i * \max(0, p_i - \max(0, t_1 - r_i) - \max(0, d_i - t_2))$$

To get an intuitive picture of the formula, notice that $\max(0, t_1 - r_i)$ is an upper bound of the number of time units during which A_i can execute before time t_1 and $\max(0, d_i - t_2)$ is an upper bound of the number of time units during which A_i can execute after time t_2. Consequently, $\max(0, p_i - \max(0, t_1 - r_i) - \max(0, d_i - t_2))$ is a lower bound of the number of time units during which A_i executes in $[t_1\ t_2]$.
We now define the overall required energy consumption $W_{PE}(t_1, t_2)$ over $[t_1\ t_2]$ as the sum over all activities A_i of $W_{PE}(A_i, t_1, t_2)$. Note that for $t_1 = t_2$, $W_{PE}(t_1, t_2)$ is defined and, under the assumption $r_i + p_i \leq d_i$, is equal to 0.

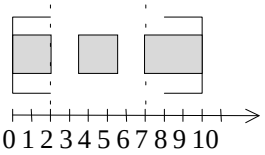


Figure 2: The required energy consumption of an activity (release date 0, deadline 10, processing time 7 and resource requirement 2) over [2 7]. The activity must execute during at least 2 time units in [2 7]; which corresponds to $W_{PE}(A, 2, 7) = 2 * (7 - (2 - 0) - (10 - 7)) = 4$

Proposition 3.
There is a feasible partially elastic schedule of I if and only if for any non-empty interval $[t_1\ t_2]$, $W_{PE}(t_1, t_2) \leq C * (t_2 - t_1)$.

Proof.
The fact that $W_{PE}(t_1, t_2) \leq C * (t_2 - t_1)$ is a necessary condition is obvious. Suppose now that there is no feasible partially elastic schedule of I. Then there is no feasible preemptive schedule of F(G(I)). Consequently, there is a set of activities S of F(G(I)) such that between

the minimum release date of activities in S and the maximum deadline of activities in S there is not enough "space" to schedule all activities in S [8].
This leads to

$$\min_{A_i^j \in S} r_i^j + \sum_{A_i^j \in S} p_i^j > \max_{A_i^j \in S} d_i^j \Rightarrow \sum p_i^j > C*(t_2 - t_1) \quad \begin{cases} r_i^j \geq C*t_1 \\ d_i^j \leq C*t_2 \end{cases}$$

where $C * t_1$ is the minimum release date in S and $C * t_2$ the maximum deadline in S (recall that release dates and deadlines of activities in S are multiple of C). Then the equation becomes:

$$\sum_i \sum_{\substack{j \text{ in } [1\ p_i] \text{ such that:} \\ C*(r_i + j - 1) \geq C*t_1 \\ C*(d_i - p_i + j) \leq C*t_2}} c_i > C * (t_2 - t_1)$$

For each i, let us now count the values of j in [1 p_i] such that (1) $C * (r_i + j - 1) \geq C * t_1$ and (2) $C * (d_i - p_i + j) \leq C * t_2$, *i.e.*, the number of integers in between $\max(1, t_1 + 1 - r_i)$ and $\min(p_i, t_2 + p_i - d_i)$. This number is equal to:

$\max(0, 1 + \min(p_i, t_2 + p_i - d_i) - \max(1, t_1 + 1 - r_i))$
$= \max(0, p_i + \min(0, t_2 - d_i) - \max(0, t_1 - r_i))$
$= \max(0, p_i - \max(0, d_i - t_2) - \max(0, t_1 - r_i))$.

Therefore, the previous equation becomes $\sum_i W_{PE}(A_i, t_1, t_2) > C * (t_2 - t_1)$. □

2.2.3 A Quadratic Algorithm

We propose a quadratic algorithm to determine whether there exists a feasible partially elastic schedule for a given instance of the Partially Elastic CuSP. This algorithm is derived from the algorithm used in [28] to compute the subset bound of the m-Machine Problem. It consists of computing the overall required energy consumption over each interval $[r_i\ d_k]$ and to test whether this energy exceeds the energy provided by the resource over this interval. We prove that such tests guarantee the existence of a feasible partially elastic schedule. To achieve this proof, we study the slack function $S_{PE}(t_1, t_2) = C * (t_2 - t_1) - W_{PE}(t_1, t_2)$.

Proposition 4.
Let t_1, t_2 be two integer values such that $t_1 < t_2$.

- If t_1 is not a release date, either $S_{PE}(t_1 + 1, t_2) < S_{PE}(t_1, t_2)$ or $S_{PE}(t_1 - 1, t_2) \leq S_{PE}(t_1, t_2)$.
- If t_2 is not a deadline, either $S_{PE}(t_1, t_2 - 1) < S_{PE}(t_1, t_2)$ or $S_{PE}(t_1, t_2 + 1) \leq S_{PE}(t_1, t_2)$.

Proof.
The two items of the proposition being symmetric, we only prove the first item.
Suppose that $S_{PE}(t_1, t_2) \leq S_{PE}(t_1 + 1, t_2)$ and $S_{PE}(t_1, t_2) < S_{PE}(t_1 - 1, t_2)$. Let us then define the sets $\Psi = \{i \mid p_i - \max(0, t_1 - r_i) - \max(0, d_i - t_2) > 0\}$ and $\Phi = \{i \mid r_i \leq t_1\}$.

- The equation $S_{PE}(t_1, t_2) \leq S_{PE}(t_1 + 1, t_2)$ can be rewritten

$$-C + \sum_{i \in \Psi} c_i * (-\max(0, t_1 - r_i) + \max(0, t_1 + 1 - r_i)) \geq 0.$$

Since $\forall i \notin \Phi$, $\max(0, t_1 - r_i) = 0$ and $\max(0, t_1 + 1 - r_i) = 0$, the previous equation becomes:

$$\sum_{i \in \Psi \cap \Phi} c_i * (-\max(0, t_1 - r_i) + \max(0, t_1 + 1 - r_i)) \geq C \Rightarrow \sum_{i \in \Psi \cap \Phi} c_i \geq C.$$

- The equation $S_{PE}(t_1, t_2) < S_{PE}(t_1 - 1, t_2)$ can be rewritten as follows:

  $\sum_i c_i * \max(0, p_i - \max(0, t_1 - 1 - r_i) - \max(0, d_i - t_2)) - \sum_i c_i * \max(0, p_i - \max(0, t_1 - r_i) - \max(0, d_i - t_2)) < C$

  $\Rightarrow \sum_{i \in \Psi} c_i * (-\max(0, t_1 - 1 - r_i) + \max(0, t_1 - r_i)) + \sum_{i \notin \Psi} c_i * \max(0, p_i - \max(0, t_1 - 1 - r_i) - \max(0, d_i - t_2)) < C$

  $\Rightarrow \sum_{i \in \Psi} c_i * (-\max(0, t_1 - 1 - r_i) + \max(0, t_1 - r_i)) < C$

  Consider now two cases.
  - ◊ If $i \in \Phi$ then $t_1 - r_i \geq 0$. Moreover, $t_1 - r_i - 1 \geq 0$ since t_1 is not a release date.
  - ◊ If $i \notin \Phi$ then $t_1 - r_i < 0$ and $t_1 - r_i - 1 < 0$.

  The previous equation then becomes $\sum_{i \in \Psi \cap \Phi} c_i < C$, which contradicts $\sum_{i \in \Psi \cap \Phi} c_i \geq C$. □

Proposition 5.

$[\forall r_j, \forall d_k > r_j, S_{PE}(r_j, d_k) \geq 0] \Leftrightarrow [\forall t_1, \forall t_2 > t_1, S_{PE}(t_1, t_2) \geq 0]$
$\Leftrightarrow$ [There exists a feasible partially elastic schedule]

Proof.

Note that if $t_1 < \min_i(r_i)$, the slack strictly increases when t_1 decreases, and if $t_2 > \max_i(d_i)$, the slack strictly increases when t_2 increases. Hence, the slack function assumes a minimal value over an interval $[t_1\ t_2]$ with $\min_i(r_i) \leq t_1 \leq t_2 \leq \max_i(d_i)$. We can assume that both t_1 and t_2 are integers (if t_1 is not, the function $t \rightarrow S_{PE}(t, t_2)$ is linear between $\lfloor t_1 \rfloor$ and $\lceil t_1 \rceil$; thus either $S_{PE}(\lfloor t_1 \rfloor, t_2) \leq S_{PE}(t_1, t_2)$ or $S_{PE}(\lceil t_1 \rceil, t_2) \leq S_{PE}(t_1, t_2)$). Among the pairs of integer values (t_1, t_2) which realize the minimum of the slack, let (u_1, u_2) be the pair such that u_1 is minimal and u_2 is maximal (given u_1).

We can suppose that $S_{PE}(u_1, u_2) < 0$ (otherwise the proposition holds). Consequently, $u_1 < u_2$ and thus, according to Proposition 4, either u_1 is a release date or $S_{PE}(u_1 + 1, u_2) < S_{PE}(u_1, u_2)$ or $S_{PE}(u_1 - 1, u_2) \leq S_{PE}(u_1, u_2)$. Since $S_{PE}(u_1, u_2)$ is minimal, the previous inequalities lead to $S_{PE}(u_1 - 1, u_2) = S_{PE}(u_1, u_2)$; which contradicts our hypothesis on u_1. Consequently, u_1 is a release date. A symmetric demonstration proves that u_2 is a deadline. □

This proposition is of great interest since it allows us to restrict the computation of W_{PE} to intervals $[t_1\ t_2]$ where t_1 is a release date and t_2 is a deadline. Before describing the algorithm, we introduce the notation $p_i^+(t_1)$ which denotes the minimal number of time units during which A_i must execute after t_1, *i.e.*, $p_i^+(t_1) = \max(0, p_i - \max(0, t_1 - r_i))$. The following algorithm computes $W_{PE}(t_1, t_2)$ over all relevant intervals.

The basic underlying idea is that, for a given t_1, the values of t_2 at which the slope of the $t \rightarrow W_{PE}(t_1, t)$ function changes are either of the form d_i or of the form $d_i - p_i^+(t_1)$. The procedure iterates on the relevant values of t_1 and t_2. Each time t_2 is modified, $W_{PE}(t_1, t_2)$ is computed, as well as the new slope (just after t_2) of the $t \rightarrow W_{PE}(t_1, t)$ function. Each time t_1 is modified, the set of activities with relevant $d_i - p_i^+(t_1)$ is incrementally recomputed and resorted.

```
1  procedure update(DP, old_t1, t1)
2  move = ∅, no_move = ∅                    // initialize two empty lists
3  for act in DP
4     if (p+act(t1) > 0) then
5        if (p+act(t1) = p+act(old_t1)) then add act to the list no_move
6        else add act to the list move
7        end if
8     end if
9  end for
10 DP = merge(move, no_move)
11
12 procedure energies
13 DD = activities sorted in increasing order of dact
14 DP = activities sorted in increasing order of dact - pact
15 old_t1 = mini(ri)
16 for t1 in the set of release dates (sorted in increasing order)
17    update(DP, old_t1, t1)
18    old_t1 = t1, iDD = 0, iDP = 0
19    W = 0, old_t2 = t1, slope = Σact WPE(act, t1, t1+1)
20    while (iDP < length(DP) or iDD < n)
21       if (iDD < n) then t2_DD = dDD[iDD + 1]
22       else t2_DD = ∞
23       end if
24       if (iDP < length(DP)) then t2_DP = dDP[iDP + 1] - p+DP[iDP + 1](t1)
25       else t2_DP = ∞
26       end if
27       t2 = min(t2_DD, t2_DP)
28       if (t2 = t2_DP) then iDP = iDP + 1, act = DP[iDP]
29       else iDD = iDD + 1, act = DD[iDD]
30       end if
31       if (t1 < t2) then
32          W = W + slope * (t2 - old_t2)
33          WPE(t1, t2) = W                  // energy over [t1 t2] is computed
34          old_t2 = t2
35          slope=slope+WPE(act,t1,t2+1)-2*WPE(act,t1,t2)+WPE(act,t1,t2-1)
36       end if
37    end while
38 end for
```

Let us detail the procedure **energies**.

- Lines 13 and 14 initialize **DD** and **DP**. **DD** is the array of activities sorted in increasing order of deadlines and **DP** is the array of activities sorted in increasing order of $d_i - p_i$.
- The main loop (line 16) consists in an iteration over all release dates t_1. Notice that **old_t1** allows to keep the previous value of t_1.
- The procedure **update(DP, old_t1, t1)** reorders the array **DP** in increasing order of $d_i - p_i^+(t_1)$. This procedure will be described later on.
- Before starting the inner loop, a variable **slope** is initialized (line 19). It corresponds to the slope of the function $t \rightarrow W_{PE}(t_1, t)$ immediately after the time point **old_t2**. **old_t2** and **slope** are initialized line 19 and updated lines 34 and 35.
- The inner loop on t_2 (lines 20-37) consists in iterating on both arrays **DD** and **DP** at the same time. Because both arrays are sorted, some simple operations (line 21 to 27) determine the next value of t_2. Notice that t_2 can take at most 2 * n values and that t_2 takes all the values which correspond to a deadline. The indices i_{DD} and i_{DP} correspond to the current position in arrays **DD** and **DP** respectively.
- Lines 28 to 30 enable to increase one of the indices and to determine the activity **act** which has induced the current iteration.
- To understand lines 31 to 36, consider the following rewriting of $W_{PE}(A_i, t_1, t_2)$.

$$\begin{aligned} W_{PE}(A_i, t_1, t_2) &= 0 && \text{if } t_2 \le d_i - p_i^+(t_1) \\ &= c_i * (t_2 - d_i + p_i^+(t_1)) && \text{if } d_i - p_i^+(t_1) < t_2 \le d_i \\ &= c_i * p_i^+(t_1) && \text{if } d_i < t_2 \end{aligned}$$

Between two consecutive values of **`t2`** in the inner loop, the function W_{PE} is linear. The required energy consumption between **`t1`** and **`old_t2`** is **`W`**, as computed at the end of the previous iteration. In addition, the slope of $t \rightarrow W_{PE}(\mathbf{t_1}, t)$ between **`old_t2`** and **`t2`** is **`slope`**. So, the required energy consumption between **`t1`** and **`t2`** is **`W + slope * (t2 - old_t2)`**. Then, **`slope`** is updated to take into account the non-linearity of the required energy consumption of activity A_i (**`act`** in the pseudo code) at time **`t2`**. Notice that the algorithm may execute several times lines 31-36 for the same values of **`t1`** and **`t2`** (*e.g.*, if $d_i = d_j$ for some i and j). In such a case, the slope is modified several times, with respect to all the activities inducing a non-linearity at time **`t2`**.

Let us now detail the procedure **`update`**. This procedure reorders the array DP in increasing order of $d_i - p_i^+(t_1)$. This is done in linear time. We rely on the fact that when we move from **`old_t1`** to **`t1`**, three cases can occur.

- Either $p_i^+(t_1)$ is null and then the required energy consumption of A_i in $[t_1\ t_2]$ is null; and A_i can be removed;
- Or $p_i^+(t_1) = p_i^+(old\_t_1)$ (line 5);
- Or $p_i^+(t_1) = p_i^+(old\_t_1) - (t_1 - old\_t_1)$ (line 6).

Activities are taken in the initial order of **`DP`** and are stored in either the list **`no_move`** (second item) or in the list **`move`** (third item). Notice that **`no_move`** is sorted in increasing order of $d_i - p_i^+(old\_t_1) = d_i - p_i^+(t_1)$. Moreover, **`move`** is sorted in increasing order of $d_i - p_i^+(old\_t_1)$ but **`move`** is also sorted in increasing order of $d_i - p_i^+(t_1)$ since the difference between $p_i^+(t_1)$ and $p_i^+(old\_t_1)$ is constant for all activities in **`move`**. This means that we only have to merge **`move`** and **`no_move`** to obtain the reordered array.

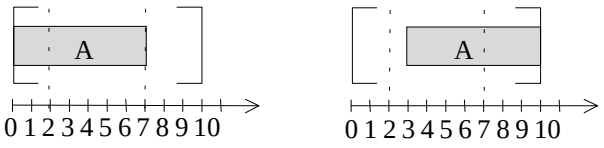
The overall algorithm runs in $O(n^2)$ since (1) the initial sort can be done in $O(n * \log(n))$, (2) the procedure **`update`** is basically a merging procedure which runs in O(n), (3) the initial value of **`slope`** for a given **`t1`** is computed in O(n), and (4) the inner and outer loops of the algorithm both consist in O(n) iterations.

2.3 A "Left-Shift/Right-Shift" Necessary Condition for the CuSP

The required energy consumption as defined in Section 2.2.2 is still valid if we consider that activities can be interrupted. In fact, [17] and [23] propose a sharper definition of the required energy consumption that takes into account the fact that activities cannot be interrupted. Given an activity A_i and a time interval $[t_1\ t_2]$, $W_{Sh}(A_i, t_1, t_2)$, the "left-shift/right-shift" required energy consumption of A_i over $[t_1\ t_2]$ is c_i times the minimum of the three following durations.

- $t_2 - t_1$, the length of the interval;
- $p_i^+(t_1) = \max(0, p_i - \max(0, t_1 - r_i))$, the number of time units during which A_i executes after time t_1 if A_i is left-shifted, *i.e.*, scheduled as soon as possible;
- $p_i^-(t_2) = \max(0, p_i - \max(0, d_i - t_2))$, the number of time units during which A_i executes before time t_2 if A_i is right-shifted, *i.e.*, scheduled as late as possible.

This leads to $W_{Sh}(A_i, t_1, t_2) = c_i * \min(t_2 - t_1, p_i^+(t_1), p_i^-(t_2))$ (see Figure 3 for an example).



Left Shift: $p_A^+(2) = 5$ Right Shift: $p_A^-(7) = 4$

Figure 3: The required energy consumption of an activity A (release date 0, deadline 10, processing time 7 and resource requirement 2) over [2 7]. At least 4 time units of A have to be executed in [2 7]; which corresponds to the formula $W_{Sh}(A, 2, 7) = 2 * \min(5, 5, 4) = 8$.

We can now define the left-shift/right-shift overall required energy consumption $W_{Sh}(t_1, t_2)$ over an interval $[t_1\ t_2]$ as the sum over all activities A_i of $W_{Sh}(A_i, t_1, t_2)$. We can also define the left-shift/right-shift slack over $[t_1\ t_2]$: $S_{Sh}(t_1, t_2) = C * (t_2 - t_1) - W_{Sh}(t_1, t_2)$. It is obvious that if there is a feasible schedule of an instance of the CuSP then, $\forall\ t_1, \forall\ t_2 \geq t_1$, $S_{Sh}(t_1, t_2) \geq 0$.

2.3.1 Characterization of relevant and irrelevant intervals

In Section 2.2 we showed that for the partially elastic relaxation, it was sufficient to calculate the slack only for those intervals $[t_1\ t_2]$ that are in the Cartesian product of the set of release dates and of the set of deadlines. In this section we show that in the left-shift/right-shift case, a larger number of intervals must be considered. On top of that, we provide a precise characterization of the set of intervals for which the slack needs to be calculated to guarantee no interval with negative slack exists.

Proposition 6.

Let us define the sets O_1, O_2 and $O(t)$.

- $O_1 = \{r_i, 1 \leq i \leq n\} \cup \{d_i - p_i, 1 \leq i \leq n\} \cup \{r_i + p_i, 1 \leq i \leq n\}$
- $O_2 = \{d_i, 1 \leq i \leq n\} \cup \{r_i + p_i, 1 \leq i \leq n\} \cup \{d_i - p_i, 1 \leq i \leq n\}$
- $O(t) = \{r_i + d_i - t, 1 \leq i \leq n\}$

$$\forall\ t_1, \forall\ t_2 \geq t_1\ S_{Sh}(t_1, t_2) \geq 0 \quad \Leftrightarrow \quad \forall\ s \in O_1, \forall\ e \in O_2, e \geq s\ , S_{Sh}(s, e) \geq 0$$
$$\text{and} \quad \forall\ s \in O_1, \forall\ e \in O(s), e \geq s, S_{Sh}(s, e) \geq 0$$
$$\text{and} \quad \forall\ e \in O_2, \forall\ s \in O(e), e \geq s, S_{Sh}(s, e) \geq 0$$

To prove Proposition 6, we first need to prove some technical properties of W_{Sh} (Propositions 7, 8 and 9). In the following, we consider that W_{Sh} is defined on $\Re^2$ and equals 0 when $t_2 \leq t_1$.

Proposition 7.

Let A_i be an activity and $(t_1, t_2) \in \Re^2$ with $t_1 < t_2$. If $t_1 \notin \{r_i, d_i - p_i, r_i + p_i\}$ and $t_2 \notin \{d_i, d_i - p_i, r_i + p_i\}$, then $\Phi(h) = W_{Sh}(A_i, t_1 + h, t_2 - h)$ is linear around 0.

Proof.

$\Phi(h)$ can be rewritten $\Phi(h) = c_i * \max(0, \min(t_2 - t_1 - 2 * h, p_i, r_i + p_i - t_1 - h, t_2 - d_i + p_i - h))$. Each of the terms $0, t_2 - t_1 - 2 * h, p_i, r_i + p_i - t_1 - h, t_2 - d_i + p_i - h$ is linear in h and if for $h = 0$, one term only realizes $\Phi(0)$, we can be sure that a small perturbation of h will have a linear effect. Assume two terms are equal and realize $\Phi(0)$. Since there are five terms, this leads us to distinguish ten cases. (1) $\Phi(0) = 0 = t_2 - t_1$, (2) $\Phi(0) = 0 = p_i$, (3) $\Phi(0) = 0 = r_i + p_i - t_1$, (4) $\Phi(0) = 0 = t_2 - d_i + p_i$, (5) $\Phi(0) = t_2 - t_1 = p_i$, (6) $\Phi(0) = t_2 - t_1 = r_i + p_i - t_1$, (7) $\Phi(0) = t_2 - t_1 = t_2 - d_i + p_i$, (8) $\Phi(0) = p_i = r_i + p_i - t_1$, (9) $\Phi(0) = p_i = t_2 - d_i + p_i$, (10) $\Phi(0) = r_i + p_i - t_1 = t_2 - d_i + p_i$. According to our hypotheses, all cases are impossible except (5) and (10).

- We claim that case (5) cannot occur. Since $t_2 - t_1 = p_i$ and since this value is equal to $\Phi(0)$, we have $p_i < r_i + p_i - t_1$ and $p_i < t_2 - d_i + p_i$ (equality cannot occur because of our hypotheses). Thus, $t_2 - t_1 = p_i > d_i - r_i$; which contradicts the fact that $r_i + p_i \leq d_i$.

- If (10) holds then $r_i + p_i - t_1 - h = t_2 - d_i + p_i - h$. We can moreover suppose that these two terms are the only ones to realize $\Phi(0)$ (otherwise one of the previous cases would occur). Around 0, $\Phi(h)$ can be rewritten $c_i * (r_i + p_i - t_1 - h)$; which is linear. □

Proposition 8.

Let A_i be an activity and $(t_1, t_2) \in \Re^2$ such that $t_1 < t_2$ and $t_2 \notin \{d_i, d_i - p_i, r_i + p_i, r_i + d_i - t_1\}$, then $\Theta(h) = W_{Sh}(A_i, t_1, t_2 - h)$ is linear around 0.

Proof.
Similar to the proof of Proposition 7 (see [5] for details). □

Proposition 9.

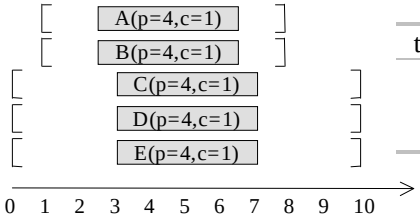
Let $(t_1, t_2) \in \Re^2$ such that $t_1 < t_2$.

- If $t_1 \notin O_1$ and $t_2 \notin O_2$, $h \rightarrow S_{Sh}(t_1 + h, t_2 - h)$ is linear around 0.
- If $t_2 \notin O_2 \cup O(t_1)$, $h \rightarrow S_{Sh}(t_1, t_2 - h)$ is linear around 0.
- If $t_1 \notin O_1 \cup O(t_2)$, $h \rightarrow S_{Sh}(t_1 + h, t_2)$ is linear around 0.

Proof (sketch).
We prove the first item. Since $t_1 \notin O_1$ and $t_2 \notin O_2$, $\forall$ i, $h \rightarrow W_{Sh}(A_i, t_1 + h, t_2 - h)$ is linear around 0 (Proposition 7). Thus, $h \rightarrow S_{Sh}(t_1 + h, t_2 - h)$ is linear around 0. The same proof applies for both other items (Proposition 8 and its symmetric counterpart are used). □

Proof of Proposition 6.
The implication from left to right is obvious. Suppose now that the right hand side of the equivalence holds and that there exists an interval $[t_1\ t_2]$ for which the slack is strictly negative. As in the partially elastic case, we remark that when t_1 is smaller than $r_{min} = \min(r_i)$, the slack strictly increases when t_1 decreases. Similarly, when t_2 is greater than $d_{max} = \max(d_i)$, the slack strictly increases when t_2 increases. Since the slack function is also continuous, it assumes a minimal value over an interval $[t_1\ t_2]$ with $r_{min} \le t_1 \le t_2 \le d_{max}$. Let us consequently select a pair (t_1, t_2) at which the slack is minimal. In case several pairs (t_1, t_2) minimize the slack, an interval with maximal length is chosen. Since this slack is strictly negative, we must have $t_1 < t_2$.
Case 1: If $t_1 \notin O_1$ and $t_2 \notin O_2$, then according to Proposition 9, $\varphi(h) = S_{Sh}(t_1 + h, t_2 - h)$ is linear around 0. Since (t_1, t_2) is a global minimum of the slack, $\varphi(h)$ is constant around 0, which contradicts the fact that the length of $[t_1\ t_2]$ is maximal.
Case 2: If $t_1 \in O_1$ then $t_2 \notin O_2 \cup O(t_1)$, otherwise the slack is non-negative. According to Proposition 9, $\theta(h) = S_{Sh}(t_1, t_2 - h)$ is linear around 0. Since (t_1, t_2) is a global minimum of the slack, $\theta(h)$ is constant around 0, which contradicts the fact that the length of $[t_1\ t_2]$ is maximal.
Case 3: If $t_2 \in O_2$ then $t_1 \notin O_1 \cup O(t_2)$, otherwise the slack is positive. Because of Proposition 9, $\theta(h) = S_{Sh}(t_1 + h, t_2)$ is linear around 0. Since (t_1, t_2) is a global minimum of the slack, $\theta(h)$ is constant around 0, which contradicts the fact that the length of $[t_1\ t_2]$ is maximal.
The combination of cases 1, 2 and 3 leads to a contradiction. This concludes the proof. □

Proposition 6 provides a characterization of interesting intervals over which the slack must be computed to ensure it is always non-negative over any interval. This characterization is weaker than the one proposed for the partially elastic case where the interesting time intervals $[t_1\ t_2]$ are in the Cartesian product of the set of the release dates and of the set of deadlines. However, there are still only $O(n^2)$ relevant pairs (t_1, t_2). Some of these pairs belong to the Cartesian product $O_1 * O_2$. The example of Figure 4 proves that some pairs do not.



$t_2 \setminus t_1$	0	1	2
8	16-4∗2-2∗3=2	14-4∗2-2∗3=0	12-3∗2-2∗3=0
9	18-4∗2-3∗3=1	16-4∗2-3∗3=-1	14-3∗2-2∗3=2
10	20-4∗2-4∗3=0	18-4∗2-3∗3=1	16-3∗2-2∗3=4

Values of $S_{Sh}(t_1, t_2)$

Figure 4: Some interesting time intervals are outside the Cartesian product $O_1 * O_2$. In this example, (resource of capacity 2 and 5 activities A, B, C, D, E), the pair (1, 9) corresponds to the minimal slack and does not belong to $\{0, 1, 4, 5, 6\} * \{4, 5, 6, 8, 10\}$. In this interval, the slack is negative, which proves that there is no feasible schedule. Notice that neither the fully elastic nor the partially elastic relaxation can trigger a contradiction.

2.3.2 A Quadratic Algorithm

We propose an $O(n^2)$ algorithm to compute the required energy consumption W_{Sh} over all interesting pairs of time points. Actually, the algorithm first computes all the relevant values taken by W_{Sh} over time intervals $[t_1\ t_2]$ with $t_1 \in O_1$, and then computes all the relevant values taken by W_{Sh} over time intervals $[t_1\ t_2]$ with $t_2 \in O_2$. The characterization obtained in the previous section ensures that all the interesting time intervals are examined in at least one of these steps. For symmetry reasons, we will only describe the first computation. It is achieved by the same type of technique than in the partially elastic case. An outer loop iterates on all values $t_1 \in O_1$ sorted in increasing order. Then, we consider the function $t \rightarrow W_{Sh}(A_i, t_1, t)$. This function is linear on the intervals delimited by the values d_i, $r_i + p_i$, $d_i - p_i$ and $r_i + d_i - t_1$. We rely on this property to incrementally maintain the slope of the function $t \rightarrow W_{Sh}(A_i, t_1, t)$.

```
1  DD = activities sorted in increasing order of d_act
2  RP = activities sorted in increasing order of r_act + p_act
3  DP = activities sorted in increasing order of d_act - p_act
4  RD = activities sorted in increasing order of r_act + d_act
5  for t1 in the set O1 (sorted in increasing order)
6     iDD = 0, iRP = 0, iDP = 0, iRD = 0
7     W = 0, old_t2 = t1, slope = Σ_act W_Sh(act, t1, t1+1)
8     while (iDD < n or iRP < n or iDP < n or iRD < n)
9        if (iDD < n) then t2_DD = d_DD[iDD + 1] else t2_DD = ∞ end if
10       if (iRP < n) then t2_RP = r_RP[iRP + 1] + p_RP[iRP + 1] else t2_RP = ∞ end if
11       if (iDP < n) then t2_DP = d_DP[iDP + 1] - p_DP[iDP + 1] else t2_DP = ∞ end if
12       if (iRD < n) then t2_RD = r_RD[iRD + 1] + d_RD[iRD + 1] - t1 else t2_RD = ∞ end if
13       t2 = min(t2_DD, t2_RP, t2_DP, t2_RD)
14       if (t2 = t2_DD) then iDD = iDD + 1, act = DD[iDD]
15       else if (t2 = t2_iRP) then iRP = iRP + 1, act = RP[iRP]
16       else if (t2 = t2_iDP) then iDP = iDP + 1, act = DP[iDP]
17       else if (t2 = t2_iRD) then iRD = iRD + 1, act = RD[iRD]
18       end if
19       if (t1 < t2) then
20          W = W + slope * (t2 - old_t2)
21          W_Sh(t1, t2) = W                    // energy over [t1 t2] is computed
22          old_t2 = t2
23          slope=slope+W_Sh(act,t1,t2+1)-2*W_Sh(act,t1,t2)+W_Sh(act,t1,t2-1)
24       end if
25    end while
26 end for
```

There are some few differences with the partially elastic case.

- The computation of t_2 is slightly more complex since there are more interesting values to consider.
- One does not need to reorder any list: when t_1 increases, none of the values d_i, $r_i + p_i$, $d_i - p_i$ and $r_i + d_i - t_1$ changes except the last one; which corresponds to the list RD. Since RD is sorted in increasing order of $r_i + d_i$, it is also sorted in increasing order of $r_i + d_i - t_1$.
- In line 23, one shall in fact be careful not to update the slope more than once for the same tuple (t_1, t_2, act). This can be done easily by marking the activity act with t_2. We have not included this marking in the pseudo-code to keep it simple.

2.4 Synthesis of Theoretical Results

Figure 5 summarizes the theoretical results related to the conditions described in Sections 2.1, 2.2 and 2.3. The most satisfactory results are obtained for the Fully Elastic CuSP and for the Partially Elastic CuSP since the related problems are polynomially solvable by either a reduction to a One-Machine Preemptive Problem or by some simple and intuitive energetic computation. Less strong results are obtained for the CuSP. No polynomial sufficient condition for the existence of a feasible schedule is proposed (which seems reasonable since the CuSP is NP-complete). As for the Partially Elastic CuSP, the left-shift/right-shift necessary condition relies on energetic reasoning, but there are more time intervals to consider in practice and the structure of these intervals is far more complex and poorly intuitive.

Needless to say, all relaxations can be used as part of the resolution of a non-preemptive cumulative problem. However, it is easy to see that the necessary condition based on the fully elastic relaxation is subsumed by the one based on the partially elastic relaxation, which in turn is subsumed by the left-shift/right-shift necessary condition.

	Fully Elastic	Partially Elastic	Left-Shift/Right-Shift
Characterization	necessary and sufficient	necessary and sufficient	necessary
Complexity	$O(n * \log(n))$	$O(n^2)$	$O(n^2)$
Method	One-Machine reduction	slack computation	slack computation
Intervals $[t_1\ t_2]$	-	$O(n^2)$ in a Cartesian product	$O(n^2)$

Figure 5: A comparison of the 3 necessary conditions

In the m-Machine case, *i.e.*, when activities require exactly one unit of the resource, the three necessary conditions can be compared to several results of the literature (these results are discussed in [28]).

First, notice that the decision variant of the **Preemptive m-Machine Problem** is polynomial and can be formulated as a maximum flow problem (see for instance [18, 28]). As shown in [28], solving this maximum flow problem leads to a worst case complexity of $O(n^3)$. The preemptive relaxation is stronger than the fully and the partially elastic relaxations. However, it is not stronger than the left-shift/right-shift necessary condition. Indeed, there exists a feasible preemptive schedule of the m-Machine instance described on Figure 4, while the left-shift/right-shift necessary condition does not hold.

A comparison can also be made between the **subset bound**, a lower bound for the optimization variant of the m-Machine Problem (see for example [8, 11, 28]) and the partially elastic relaxation. An instance of the optimization variant of the m-Machine Problem consists of a set of activities characterized by a release date r_i, a tail q_i and a processing time p_i, and a

resource of capacity C = m. The objective is to find a start time assignment s_i for each activity A_i such that (1) temporal and resource constraints are met and (2) $\max_i (s_i + p_i + q_i)$ is minimal.

The subset bound is the maximum among all subsets J of at least C activities of the following expression, in which R(J) and Q(J) denote the sets of activities in J having respectively the C smallest release dates and the C smallest tails.

$$\frac{1}{C}\left(\sum_{A_j \in R(J)} r_j + \sum_{A_j \in J} p_j + \sum_{A_j \in Q(J)} q_j \right)$$

Perregaard [28] presents an algorithm to compute the subset bound in a quadratic number of steps. Carlier and Pinson [11] describe an $O(n * \log(n) + n * C * \log(C))$ algorithm which relies on a "pseudo-preemptive" relaxation of the m-Machine Problem. Notice that the subset bound can apply, thanks to a simple transformation, as a necessary condition of existence for the decision variant of the m-Machine Problem:

$$\forall J \subseteq \{A_1, \cdots, A_n\} \text{ such that } |J| \geq C, \sum_{A_j \in R(J)} r_j + \sum_{A_j \in J} p_j \leq \sum_{A_j \in D(J)} d_j$$

where D(J) denotes the set of activities in J having the C largest deadlines.

Proposition 10.
In the m-Machine case, there exists a feasible partially elastic schedule if and only if the subset bound necessary condition holds.

Proof.

- First, assume that there exists a feasible partially elastic schedule. Let J be any subset of at least C activities. Let $\rho_1, \rho_2, ..., \rho_C$ be the C smallest release dates of activities in J, and $\delta_1, \delta_2, ..., \delta_C$ be the C largest deadlines of activities in J. Since before ρ_C, at most C activities execute, and since for each of these activities A_i at most $(\rho_C - r_i)$ units are executed before ρ_C, the schedule of these activities can be reorganized so that pes(t, i) is at most 1 for every $t \leq \rho_C$. Let us now replace each activity A_i in R(J) with a new activity of release date ρ_1, deadline d_i, and duration $p_i + (r_i - \rho_1)$. A feasible partially elastic schedule of the new set of activities is obtained as a simple modification of the previous schedule, by setting pes(t, i) = 1 for every A_i in R(J) and t in $[\rho_1\ r_i)$. The same operation can be done between δ_C and δ_1 for the activities in D(J). As a result, we have a partially elastic schedule requiring $\Sigma_{A_i \in J} p_i + \Sigma_{A_i \in R(J)}(r_i - \rho_1) + \Sigma_{A_i \in D(J)}(\delta_1 - d_i)$ units of energy between ρ_1 and δ_1. Hence, we have $\Sigma_{A_i \in J} p_i + \Sigma_{A_i \in R(J)}(r_i - \rho_1) + \Sigma_{A_i \in D(J)}(\delta_1 - d_i) \leq C * (\delta_1 - \rho_1)$, which is equivalent to the subset bound condition for J.
- We now demonstrate the other implication. Assume that the slack S_{PE} is strictly negative for some interval. Let then $[t_1\ t_2]$ be the interval over which the slack is minimal and let us define J as the set of activities A_i such that $W_{PE}(A_i, t_1, t_2) > 0$. Notice that there are at least C activities in J because $r_i + p_i \leq d_i$ implies that $W_{PE}(A_i, t_1, t_2) \leq t_2 - t_1$. In addition, at most C activities A_i in J are such that $r_i < t_1$. Otherwise, when t_1 is replaced by $t_1 - 1$, the slack decreases. Similarly, there are at most C activities A_i in J are such that $t_2 < d_i$. Let us define $X = \{A_i \in J \mid r_i < t_1\}$ and $Y = \{A_i \in J \mid t_2 < d_i\}$. According to the previous remark, we have $|X| \leq C$ and $|Y| \leq C$. Now notice that for any activity A_i in J, $W_{PE}(A_i, t_1, t_2) > 0$. Thus, we have $W_{PE}(A_i, t_1, t_2) = \Sigma_{A_i \in J}(p_i - \max(0, t_1 - r_i) - \max(0, d_i - t_2))$. This can be rewritten:

$$W_{PE}(A_i, t_1, t_2) = \Sigma_{A_i \in J} p_i + \Sigma_{A_i \in X} r_i - |X| * t_1 - \Sigma_{A_i \in Y} d_i + |Y| * t_2.$$

Since $S_{PE}(t_1, t_2)$ is strictly negative, we have

$$\Sigma_{A_i \in X} r_i + (C - |X|) * t_1 + \Sigma_{A_i \in J} p_i > \Sigma_{A_i \in Y} d_i + (C - |Y|) * t_2.$$

Because of the definition of R(J) (respectively D(J)), and because $|X| \leq C$ and $|Y| \leq C$, we have $\Sigma_{A_i \in R(J)} r_j \geq \Sigma_{A_i \in X} r_j + (C - |X|) * t_1$ and $\Sigma_{A_i \in D(J)} d_j \leq \Sigma_{A_i \in Y} d_j + (C - |Y|) * t_2$. As a consequence, $\Sigma_{A_i \in R(J)} r_j + \Sigma_{A_i \in J} p_i > \Sigma_{A_i \in D(J)} d_j$, which is exactly the subset bound necessary condition for the set J. □

3 Time-Bound Adjustments for the CuSP

This section describes three time-bound adjustment schemes for the CuSP. These techniques extend the time-bound adjustments, also called edge-finding, initially proposed for the One-Machine Problem [2, 3, 9, 10, 12, 25, 29].

3.1 Time-Bound Adjustments for the Fully Elastic CuSP

In the fully elastic case, we rely on the reduction of the Fully Elastic CuSP to the Preemptive One-Machine Problem as described in Section 2.1. We then use the time-bound adjustment algorithm proposed in [22] for the Preemptive One-Machine Problem.
More precisely, the adjustment scheme is:

1. Build the One-Machine Preemptive Problem instance F(I) corresponding to the Fully Elastic CuSP instance I.
2. Apply the preemptive edge-finding algorithm on activities A_1', ..., A_n' of the instance F(I). As explained in [22], for each activity A_i', four time-bounds can be sharpened: the release date r_i', the latest possible start time LST_i', the earliest possible end time EET_i', and the deadline d_i'.
3. Update the time-bounds of each A_i. $r_i = \lfloor r_i' / C \rfloor$, $LST_i = \lfloor LST_i' / C \rfloor$, $EET_i = \lceil EET_i' / C \rceil$ and $d_i = \lceil d_i' / C \rceil$.

This algorithm runs in a quadratic number of steps since (1) and (3) are linear and (2) can be done in $O(n^2)$ as detailed in [22].

Proposition 11.
The time-bound adjustments made by the algorithm above are the best possible ones, *i.e.*, the lower and upper bounds for the start and end time of activities can be reached by some feasible fully elastic schedules.

Proof (sketch).
The same proof applies for each of the four time-bounds. We focus here on the earliest end time. The basic idea is to prove that for any activity A_i, (1) there is a fully elastic schedule on which A_i can end at the earliest end time computed by the fully elastic time-bound adjustment algorithm and (2) there is no fully elastic schedule on which A_i can end before the earliest end time computed by the algorithm. Both steps can be proven thanks to the reduction F, and to the fact that the preemptive edge-finding algorithm computes the best possible time-bounds for the Preemptive One-Machine Problem [22]. □

3.2 Time-Bound Adjustments for the CuSP Based on the Partially Elastic Relaxation

In this section, we provide an adjustment scheme for the CuSP which relies on the required energy consumptions computed in the partially elastic case. From now on, we assume that $\forall\ r_j, \forall\ d_k, W_{PE}(r_j, d_k) \leq C * (d_k - r_j)$. If not, we know that there is no feasible partially elastic schedule. As for other adjustment techniques, our basic idea is to try to order activities. More precisely, given an activity A_i and an activity A_k, we examine whether A_i can end before d_k.

Proposition 12.[1]
If $\exists\ A_j \mid r_j < d_k$ and $W_{PE}(r_j, d_k) - W_{PE}(A_i, r_j, d_k) + c_i * \max(0, p_i - \max(0, r_j - r_i)) > C * (d_k - r_j)$ then a valid lower bound of the end time of A_i is:

$$d_k + \frac{W_{PE}(r_j,d_k) - W_{PE}(A_i,r_j,d_k) + c_i * \max(0, p_i - \max(0, r_j - r_i)) - C*(d_k - r_j)}{c_i}$$

Proof.
Notice that $W_{PE}(r_j, d_k) - W_{PE}(A_i, r_j, d_k) + c_i * \max(0, p_i - \max(0, r_j - r_i))$ is the overall required energy consumption over $[r_j\ d_k]$ when d_i is set to d_k. If this quantity is greater than $C * (d_k - r_j)$ then A_i must end after d_k. To understand the lower bound of the end time of A_i, simply notice that the numerator of the expression is the number of energy units of A_i which have to be shifted after time d_k. We can divide this number of units by the amount of resource required by A_i to obtain a lower bound of the duration required to execute these units. □

As all values $W_{PE}(r_j, d_k)$ can be computed in $O(n^2)$, this mechanism leads to a simple $O(n^3)$ algorithm. For any tuple (A_i, A_j, A_k) (1) check if A_i can end before d_k and (2) in such a case compute the corresponding time-bound adjustment. The issue is that $O(n^3)$ is a high complexity. This led us to further investigation. In the following, we show that the same adjustments can be made in $O(n^2 * |\{c_i\}|)$, through successive transformations and decompositions of the adjustment scheme.
Given A_i and A_k our goal is to find the activity A_j which will produce the best possible adjustment. Notice that if $d_i \leq d_k$, $W_{PE}(A_i, r_j, d_k) = c_i * \max(0, p_i - \max(0, r_j - r_i))$ and then no adjustment can be achieved since we assumed that $\forall\ r_j, \forall\ d_k, W_{PE}(r_j, d_k) \leq C * (d_k - r_j)$. In the following, we only consider the case in which $d_i > d_k$. This can be written as a mathematical optimization problem.

$$\max_j \left(d_k + \frac{W_{PE}(r_j,d_k) - W_{PE}(A_i,r_j,d_k) + c_i * \max(0, p_i - \max(0, r_j - r_i)) - C*(d_k - r_j)}{c_i} \right) \quad \text{u.c.} \begin{cases} W_{PE}(r_j,d_k) - W_{PE}(A_i,r_j,d_k) + c_i * \max(0, p_i - \max(0, r_j - r_i)) > C*(d_k - r_j) \\ r_j < d_k \end{cases} \quad (P)$$

Let $W_{j,k} = W_{PE}(r_j, d_k)$ if $r_j < d_k$, and $-\infty$ otherwise. Note that the $W_{j,k}$ can be pre-computed in $O(n^2)$ as shown in Section 2.2.3. Then, P can be reduced to the following problem.

---

[1] Note that the lower bound proposed in this proposition only holds for the CuSP and does not hold for the Partially Elastic CuSP. Indeed, in the partially elastic case, nothing prevents A_i from using more than c_i units of the resource at a given time point. To get a valid lower bound, one has to divide the numerator by C instead of c_i. In practice, we use only the Partially Elastic CuSP as a relaxation of the CuSP, which justifies the use of c_i to get a better bound.

$$\max_j \left(W_{j,k} + c_i * \left(\max(0,p_i - \max(0,r_j - r_i)) - \max(0,p_i - \max(0,r_j - r_i) - \max(0,d_i - d_k))\right) + C * r_j\right)$$

$$\text{u.c. } W_{j,k} + c_i * \left(\max(0,p_i - \max(0,r_j - r_i)) - \max(0,p_i - \max(0,r_j - r_i) - \max(0,d_i - d_k))\right) + C * r_j > C * d_k$$

As $C * d_k$ does not depend on j, we can first compute the maximum of the expression $W_{j,k} + c_i * (\max(0, p_i - \max(0, r_j - r_i)) - \max(0, p_i - \max(0, r_j - r_i) - \max(0, d_i - d_k))) + C * r_j$ and then check whether it is greater than $C * d_k$.

For all j such that $p_i \leq \max(0, r_j - r_i)$, $p_i - \max(0, r_j - r_i)$ is smaller than or equal to $\max(0, p_i - \max(0, r_j - r_i))$ and $f(j) = W_{j,k} + C * r_j$ which, under our hypothesis does not exceed $C * d_k$. Consequently, we can replace $\max(0, p_i - \max(0, r_j - r_i))$ by $p_i - \max(0, r_j - r_i)$ in the expression above.

As a result, we seek to maximize $W_{j,k} + c_i * (p_i - \max(0, r_j - r_i) - \max(0, p_i - \max(0, r_j - r_i) - d_i + d_k)) + C * r_j$ which is equivalent to $\max_j(W_{j,k} + c_i * (p_i - \max(0, r_j - r_i, p_i - d_i + d_k)) + C * r_j)$.

This problem can be split into two sub-problems Π_1 and Π_2 by adding respectively the constraint $r_j \leq r_i + \max(0, p_i - d_i + d_k)$ and the constraint $r_j \geq r_i + \max(0, p_i - d_i + d_k)$. Indeed, an optimal solution j^* of the original problem is either an optimal solution j_1^* for Π_1 or an optimal solution j_2^* for Π_2.

$$\max_j \left(W_{j,k} + C * r_j\right) \quad \text{u.c. } r_j \leq r_i + \max(0, p_i - d_i + d_k) \qquad (\Pi_1)$$

$$\max_j \left(W_{j,k} + (C - c_i) * r_j\right) \quad \text{u.c. } r_j \geq r_i + \max(0, p_i - d_i + d_k) \qquad (\Pi_2)$$

3.2.1 Resolution of Π_i for all i

We propose an $O(n^2)$ algorithm to compute the optima of Π_1 for all pairs of activities (A_i, A_k). The basic idea consists of rewriting the constraint of Π_1. Indeed, $r_j \leq r_i + \max(0, p_i - d_i + d_k)$ is equivalent to $r_j \leq r_{f(k,i)}$, where $A_{f(k,i)}$ is the activity with the largest release date such that either $r_{f(k,i)} \leq r_i$ or $r_{f(k,i)} \leq r_i + p_i - d_i + d_k$. Let now $\Omega_{k,u}$ denote the optimum of $W_{j,k} + C * r_j$ under the constraint $r_j \leq r_u$, then the optimum of Π_1 is $\Omega_{k,f(k,i)}$. This rewriting is of great interest since computing the values of $\Omega_{k,u}$ and $f(k, i)$ can be done in linear time for a given k.

```
1  R = activities sorted in increasing order of r_act
2  RPD = activities sorted in increasing order of r_act + p_act - d_act
3  for any activity act_k
4     let Ω be an array of integers (by convention Ω[0] = -∞)
5     for iR = 1 to iR = n
6        Ω[iR] = max(Ω[iR - 1], W_PE(r_R[iR], d_act_k) + C * r_R[iR])
7     end for
8     for iR = 1 to iR = n
9        f_R[iR] = iR
10    end for
11    iR = 0, iRPD = 0
12    while (iR < n or iRPD < n)
13       if (iR < n) then Rval = r_R[iR + 1]
14       else Rval = ∞ end if
15       if (iRPD < n) then RPDval = r_RPD[iRPD + 1] + p_RPD[iRPD + 1] - d_RPD[iRPD + 1] + d_act_k
16       else RPDval = ∞ end if
17       if (Rval ≤ RPDval) then
18          iR = iR + 1
19       else
20          iRPD = iRPD + 1
21          f_RPD[iRPD] = max(f_RPD[iRPD], iR)
22       end if
23    end while
24 end for
```

Let us comment the algorithm.

- Lines 1 and 2 achieve two initial sorts of activities. Notice that since **RPD** is sorted in increasing order of $r_i + p_i - d_i$, it is also sorted in increasing order of $r_i + p_i - d_i + d_k$ for any value of d_k.
- Lines 4 to 7 compute the values of Ω. We rely on the fact that activities are taken in increasing order of release dates and thus, the recurrence property of line 6 holds.
- Lines 11 to 23 compute the values of f(k, i). Since k is fixed, we use the notation f_{A_i} to denote the index of the activity in **R** such that its release date is $r_{f(k, i)}$. Since $A_{f(k, i)}$ is the activity with the largest release date such that either $r_{f(k, i)} \leq r_i$ or $r_{f(k, i)} \leq r_i + p_i - d_i + d_k$, we first initialize **`f`$_{\text{R[iR]}}$ `= iR`** (lines 8 to 10) and then iterate over both lists **R** and **RPD** at the same time. **Rval** and **RPDval** correspond respectively to the release date of the current activity in R and to the "$r_i + p_i - d_i + d_k$" of the current activity in **RPD**. If it comes that **Rval > RPDval** then the value of **`f`$_{\text{RPD[iRPD]}}$** can be updated (line 21) since there the release date of the **`iR`**th activity in **R** has the largest release date lower than or equal to $r_i + p_i - d_i + d_k$.

Notice that this algorithm runs in $O(n^2)$ since it consists in two initial sorts and in one outer loop over the n activities and one inner loop (2 * n iterations).

3.2.2 Resolution of Π_2 for all i

For each different value λ of c_i, the problem Π_2 can be solved with a similar algorithm. C is replaced by $C - \lambda$ which is taken as a constant and the optima for all pairs of activities (A_i, A_k) with $c_i = \lambda$ are computed (with a simple transformation needed to accommodate the direction of the inequality $r_j \geq r_i + \max(0, p_i - d_i + d_k)$). We then have an $O(n^2 * |\{c_i\}|)$ algorithm to solve Π_2. Notice that in the m-Machine case, this reduces to a simple $O(n^2)$.

In [5], we present an algorithm that solves Π_2, and hence perform the time-bound adjustments, in $O(n^2 * \log(|\{c_i\}|))$. However, this algorithm requires the use of complex data structures and is out of the scope of this paper.

3.3 Time-Bound Adjustments based on "Left-Shift/Right-Shift" for the CuSP

As in Section 3.2, the values of W_{Sh} can be used to adjust time-bounds. Given an activity A_i and a time interval $[t_1\ t_2]$ with $t_2 < d_i$, we examine whether A_i can end before t_2.

Proposition 13.

If $\exists\ t_1 \mid t_1 < t_2$ and $W_{Sh}(t_1, t_2) - W_{Sh}(A_i, t_1, t_2) + c_i * \min(t_2 - t_1, p_i^+(t_1)) > C * (t_2 - t_1)$ then a valid lower bound of the end time of A_i is:

$$t_2 + \frac{W_{Sh}(t_1,t_2) - W_{Sh}(A_i,t_1,t_2) + c_i * \min(t_2 - t_1, p_i^+(t_1)) - C*(t_2 - t_1)}{c_i}$$

Proof.

Similar to proof of Proposition 12. □

There is an obvious $O(n^3)$ algorithm to compute all the adjustments which can be obtained on the intervals $[t_1\ t_2]$ which correspond to potential local minima of the slack function. There are $O(n^2)$ intervals of interest and n activities which can be adjusted. Given an interval and an activity, the adjustment procedure runs in O(1). The overall complexity of the algorithm is then $O(n^3)$. In spite of our efforts, we were unable to exhibit a quadratic algorithm to compute all the adjustments on the $O(n^2)$ intervals under consideration.

3.4 Synthesis of Theoretical Results

Figure 6 summarizes the theoretical results related to the time-bound adjustment techniques presented in Sections 3.1, 3.2, and 3.3. The most satisfactory result is obtained for the Fully Elastic CuSP. Indeed, time-bound adjustments for the Fully Elastic CuSP are perfectly characterized (Proposition 11).
It is easy to see that the deductions made by the fully elastic techniques (both necessary condition and time-bound adjustment) are subsumed by partially elastic techniques which are in turn subsumed by left-shift/right-shift techniques. However, the weaker the relaxation is the cheaper the complexity is. In Section 4, we evaluate the tradeoff between the quality of the deductions and their cost.

	Fully Elastic	Partially Elastic	Left-Shift/Right-Shift
Characterization	perfect	-	-
Complexity	$O(n^2)$	$O(n^2 * \|\{c_i\}\|)$	$O(n^3)$
Method	One-Machine reduction	slack computation	slack computation

Figure 6: A brief comparison of the 3 adjustment techniques

4 An Experimental Study of the Necessary Conditions and of the Adjustments

To test the algorithms proposed in Sections 2 and 3 we decided to apply them at each node of a branch and bound procedure for the Resource-Constrained Project Scheduling Problem. This procedure, a variant of [4], is described in [5]. It basically relies on the use of redundant resource constraints [4, 7] and on some simple dominance properties [4].

We kept one point of flexibility in our algorithm, which corresponds to the use of a necessary condition (Section 2) and of a time-bound adjustment scheme (Section 3).

- The first version, NO, uses none of the necessary conditions and time-bound adjustment techniques presented in this paper.
- The second version, FE, relies on the fully elastic relaxation, both for the necessary condition for existence and for the time-bound adjustments (Sections 2.1 and 3.1).
- The third version, PE, relies on the partially elastic relaxation, both for the necessary condition for existence and for the time-bound adjustments (Sections 2.2 and 3.2). However, the results reported below rely on a straightforward $O(n^3)$ implementation.
- The fourth version, LSRS, relies on the necessary condition and the time-bound adjustments based on the left-shift/right-shift energy consumption (Sections 2.3 and 3.3). However, we quickly found out that this version was too time-consuming for producing any useful result. This is quite understandable. Indeed, the number of intervals to consider is multiplied by $3*3+3+3=15$ in comparison to the partially elastic case! Hence, we tried to determine a better tradeoff, *i.e.*, to reduce the number of intervals to examine without compromising too much with respect to the effectiveness of the evaluation (detection of impossibilities and time-bound adjustments). After a few trials, we decided to reduce the set of intervals to the Cartesian product of $O_1' = \{r_i, 1 \le i \le n\} \cup \{d_i - p_i, 1 \le i \le n\}$ and $O_2' = \{d_i, 1 \le i \le n\} \cup \{r_i + p_i, 1 \le i \le n\}$.

The four versions were tested on four sets of data.

- The "Alvarez" set [1], which includes 48 instances with 27 activities, 48 instances with 51 activities and 48 instances with 103 activities (the last 48 instances are ignored in this computational study),
- The "Patterson" set [27], which includes 110 instances with 14 to 51 activities,
- The KSD set of Kolisch, Sprecher and Drexel [21], which includes 480 instances with 30 activities and 4 resources,
- The BL set [4], which includes 40 instances with either 20 or 25 activities, and 3 resources.

These sets of instances are very different one from another. We use the disjunction ratio [4] as an indicator for each instance. The disjunction ratio of an instance is the ratio between a lower bound of the number of pairs of activities which cannot execute in parallel and the overall number of pairs of distinct activities. A simple lower bound of the number of pairs of activities which cannot execute in parallel can be obtained by considering pairs {A, B} such that either there is a chain of precedence constraints between A and B, or there is a resource constraint which is violated if A and B overlap in time. As shown in [4], cumulative time-bound adjustment algorithms are more likely to be useful for highly cumulative problems (*i.e.*, problems with a low disjunctive ratio), while for highly disjunctive problems, disjunctive scheduling techniques are more likely to be useful.

The average disjunctive ratios for the four sets of instances are respectively 0.82 for the Alvarez set, 0.67 for the Patterson set, 0.56 for the KSD set and 0.33 for the BL set. As a matter of fact, the BL instances have been generated because it appeared that most of the classical instances from the literature are highly disjunctive (which is not representative of the real-life RCPSP instances that we observed at Bouygues).

Each version of our branch and bound algorithm was implemented in CLAIRE [14] and applied to each of the 726 instances, with a maximal CPU time of half an hour on a PC Dell OptiPlex GX Pro 200 MHz running Windows NT. Tables 1, 2, 3 and 4 present the results obtained respectively on the Alvarez instances, the Patterson instances, the KSD instances and the BL instances. For each version of the algorithm, each table provides the number of instances solved (including proof of optimality), the average number of backtracks to solve these instances, and the corresponding average CPU time, in seconds. For each set of instances, the two last columns of the corresponding table provide the average number of backtracks and CPU time obtained on the subset of instances solved by all of the four algorithms.

Algorithm	Number of solved instances	Backtracks solved instances	CPU time solved instances	Backtracks 73 instances	CPU time 73 instances
NO	80	4027	78.1	244	11.2
FE	78	1810	58.4	244	14.3
PE	76	990	88.1	244	39.1
LSRS	73	243	64.7	243	64.7

Table 1: Experimental results on the 96 Alvarez instances (27 or 51 activities)

Algorithm	Number of solved instances	Backtracks solved instances	CPU time solved instances	Backtracks 110 instances	CPU time 110 instances
NO	110	143	1.6	143	1.6
FE	110	139	2.3	139	2.3
PE	110	128	7.7	128	7.7
LSRS	110	111	8.3	111	8.3

Table 2: Experimental results on the 110 Patterson instances

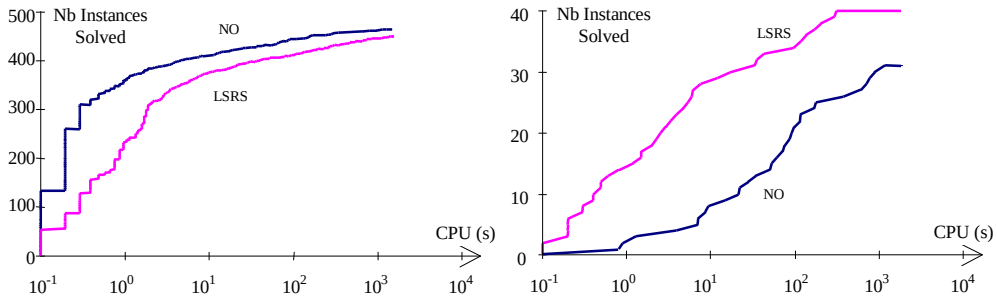
Algorithm	Number of solved instances	Backtracks solved instances	CPU time solved instances	Backtracks 451 instances	CPU time 451 instances
NO	465	4612	26.1	1181	6.8
FE	461	2593	28.9	1101	12.8
PE	453	1455	52.9	909	47.2
LSRS	451	794	52.8	794	52.8

Table 3: Experimental results on the 480 KSD instances

Algorithm	Number of solved instances	Backtracks solved instances	CPU time solved instances	Backtracks 31 instances	CPU time 31 instances
NO	31	84632	187.9	84632	187.9
FE	39	17171	80.9	4839	27.1
PE	40	3757	46.7	2177	30.4
LSRS	40	3400	38.4	1868	25.6

Table 4: Experimental results on the 40 BL instances

The effect of the different satisfiability tests and time-bound adjustment algorithms clearly depends on the set of instances. Considering only the instances solved by all algorithms, the reduction in the average number of backtracks between NO and LSRS is almost null on the Alvarez set, and equal to 22%, 33% and 98 % on the Patterson, KSD and BL sets (respectively). On the Alvarez, Patterson and KSD sets, the cost of the more complex time-bound adjustment algorithms is not balanced by the subsequent reduction of search, and the CPU time increases. On the contrary, LSRS performs much better than NO on the BL set. On the 31 instances solved by all algorithms, the number of backtracks is divided by 45, and the overall CPU time by more than 7. Figure 7 illustrates the behavior of NO and LSRS on the KSD instances and on the BL instances.



KSD set (480 instances) BL set (40 instances)

Figure 7: The behavior of LSRS and NO on the KSD and the BL instances. Each curve shows the number of instances solved in a given amount of CPU time.

5 Conclusion and Perspectives

In this paper, we have presented three necessary conditions for the existence of a feasible schedule for a given instance of the Cumulative Scheduling Problem, and three deductive algorithms to adjust the time-bounds of activities. Two of the three proposed techniques correspond to well-defined relaxations of the Cumulative Scheduling Problem: the fully elastic

relaxation and the partially elastic relaxation. These techniques can be used not only for standard scheduling problems, but also for preemptive scheduling problems. In addition, the fully elastic relaxation also applies when an activity requires an amount of resource capacity that is not fixed (*e.g.*, because the same activity can be done either by 2 people in 3 days or by 3 people in 2 days) or even allowed to vary over time (*e.g.*, 2 people on day 1 and 4 people on day 2). The third technique (left-shift/right-shift), which is the most powerful but also the most expensive, applies only to "non-elastic non-preemptive problems." Notice that the satisfiability tests are such that they provide the same answers when an activity A_i, which requires c_i units of the resource, is replaced by c_i activities A_i^j, each of which requires one unit of the resource. This is not true for the time-bound adjustments.

The techniques presented in this paper prove to be effective on some, but not all problem instances in the cumulative scheduling class. Computational results have shown that, on "highly disjunctive" project scheduling instances, the algorithms presented in this paper induce an overhead that is not balanced by the resulting reduction of search. On the other hand, the most expensive techniques prove to be highly useful for the resolution of less highly disjunctive problems.

Several questions are still open at this point.

- First, for the left-shift/right-shift technique, we have shown that the energetic tests can be limited to $O(n^2)$ time intervals. We have also provided a precise characterization of these intervals. However, it could be that this characterization can be sharpened in order to eliminate some intervals and reduce the practical complexity of the corresponding algorithm.
- Second, it seems reasonable to think that our time-bound adjustments could be sharpened. Even though the energetic tests can be limited (without any loss) to a given set of intervals, it could be that the corresponding adjustment rules cannot. A related open question is whether the time-bound adjustment schemes proposed in this paper subsume the rules already presented in [13, 23, 25, 26].
- Third, we have not incorporated in our branch and bound procedure all the results obtained by other researchers for the Resource-Constrained Project Scheduling Problem (RCPSP). In particular, we have not used, until now, any "intelligent backtracking" rule such as the cut-set rule of Demeulemeester and Herroelen [15]. This may seem a little "strange" given the excellent results reported in [16], in particular on the KSD instances, even with a limited use of the cut-set rule. However, it appears that many industrial scheduling problems include several additional features (including, for example, elastic activities [13]) which seem to require the use of other techniques. Nevertheless, in the case of the pure RCPSP, it would be interesting to determine how subsequent improvements of our procedure would influence experimental results, and hence our conclusions on the usefulness of the various adjustment techniques developed in this paper.

Acknowledgments

The authors would like to thank Jacques Carlier, Yves Caseau, François Laburthe, Pierre Lopez, Emmanuel Néron, Eric Pinson, and Jérôme Rogerie for many enlightening discussions on preemptive and non-preemptive cumulative scheduling. We also thank both anonymous referees for their comments and suggestions which allowed us to shorten several proofs and to clarify the overall presentation of the theory.

References

[1] R. Alvarez-Valdès and J.M. Tamarit, *Heuristic Algorithms for Resource-Constrained Project Scheduling: A Review and an Empirical Analysis*, Chapter 5 in Advances in Project Scheduling, R. Slowinski and J. Weglarz editors, Elsevier, 1989.

[2] D. Applegate and W. Cook, *A Computational Study of the Job-Shop Scheduling Problem*, ORSA Journal on Computing 3 (1991) 149-156.

[3] Ph. Baptiste and C. Le Pape, *A Theoretical and Experimental Comparison of Constraint Propagation Techniques for Disjunctive Scheduling*, Proc. 14th International Joint Conference on Artificial Intelligence, 1995.

[4] Ph. Baptiste and C. Le Pape, *Constraint Propagation and Decomposition Techniques for Highly Disjunctive and Highly Cumulative Project Scheduling Problems*, Proc. 3rd International Conference on Principles and Practice of Constraint Programming, 1997.

[5] Ph. Baptiste, C. Le Pape and W. P. M. Nuijten, *Satisfiability Tests and Time-Bound Adjustments for Cumulative Scheduling Problems*, Technical Report 98/97, Université de Technologie de Compiègne, 1998.

[6] P. Brucker and O. Thiele, *A Branch and Bound Method for the General-Shop Problem with Sequence-Dependent Setup Times*, OR Spektrum, 18 (1996) 145-161.

[7] P. Brucker, S. Knust, A. Schoo and O. Thiele, *A Branch and Bound Algorithm for the Resource-Constrained Project Scheduling Problem*, Working Paper, University of Osnabrück, 1997.

[8] J. Carlier, *Problèmes d'ordonnancement à contraintes de ressources : algorithmes et complexité*, Thèse de Doctorat d'Etat, University Paris VI, 1984.

[9] J. Carlier and E. Pinson, *A Practical Use of Jackson's Preemptive Schedule for Solving the Job-Shop Problem*, Annals of Operations Research, 26 (1990) 269-287.

[10] J. Carlier and E. Pinson, *Adjustment of Heads and Tails for the Job-Shop Problem*, European Journal of Operational Research 78 (1994) 146-161.

[11] J. Carlier and E. Pinson, *Jackson's Pseudo-Preemptive Schedule for the Pm/ri,qi/Cmax Scheduling Problem*, Technical Report 96/01, Université de Technologie de Compiègne, 1996.

[12] Y. Caseau and F. Laburthe, *Improved CLP Scheduling with Task Intervals*, Proc. 11th International Conference on Logic Programming, 1994.

[13] Y. Caseau and F. Laburthe, *Cumulative Scheduling with Task Intervals*, Proc. Joint International Conference and Symposium on Logic Programming, 1996.

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

[15] E. Demeulemeester and W. Herroelen, *A Branch-and-Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem*, Management Science 38 (1992) 1803-1818.

[16] E. Demeulemeester and W. Herroelen, *New Benchmark Results for the Resource-Constrained Project Scheduling Problem*, Technical Report, Katholieke Universiteit Leuven, 1995.

[17] J. Erschler, P. Lopez and C. Thuriot, *Raisonnement temporel sous contraintes de ressource et problèmes d'ordonnancement*, Revue d'Intelligence Artificielle 5 (1991) 7-32.

[18] A. Federgruen and H. Groenevelt, *Preemptive Scheduling of Uniform Machines by Ordinary Network Flow Techniques*, Management Science 32 (1986) 341-349.

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

[20] M. Gondran and M. Minoux, *Graphs and Algorithms*, John Wiley and Sons, 1984.

[21] R. Kolisch, A. Sprecher and A. Drexl, *Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems*, Management Science 41 (1995) 1693-1703.

[22] C. Le Pape and Ph. Baptiste, *Constraint Propagation Techniques for Disjunctive Scheduling: The Preemptive Case*, Proc. 12th European Conference on Artificial Intelligence, 1996.

[23] P. Lopez, J. Erschler and P. Esquirol, *Ordonnancement de tâches sous contraintes : une approche énergétique*, RAIRO Automatique, Productique, Informatique Industrielle 26 (1992) 453-481.

[24] P. Martin and D. B. Shmoys, *A New Approach to Computing Optimal Schedules for the Job-Shop Scheduling Problem*, Proc. 5th Conference on Integer Programming and Combinatorial Optimization, 1996.

[25] W. P. M. Nuijten, *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach*, PhD Thesis, Eindhoven University of Technology, 1994.

[26] W. P. M. Nuijten and E. H. L. Aarts, *A Computational Study of Constraint Satisfaction for Multiple Capacitated Job-Shop Scheduling*, European Journal of Operational Research 90 (1996) 269-284.

[27] J. H. Patterson, *A Comparison of Exact Approaches for Solving the Multiple Constrained Resource Project Scheduling Problem*, Management Science 30 (1984) 854-867.

[28] M. Perregaard, *Branch and Bound Methods for the Multi-Processor Job-Shop and Flow-Shop Scheduling Problem*, MSc Thesis, University of Copenhagen, 1995.

[29] E. Pinson, *Le problème de job-shop*, PhD Thesis, University Paris VI, 1988.