Horizontally Elastic Not-First/Not-Last Filtering Algorithm For Cumulative Resource Constraint *

Roger Kameugne[1,2], Sévérine Betmbe Fetgo[3], Vincent Gingras[4], Yanick Ouellet[4], and Claude-Guy Quimper[4]

[1] University of Maroua-Cameroon
[2] University of Bamenda - Cameroon
[3] University of Dschang-Cameroon
[4] Université Laval, Québec, QC, Canada

rkameugne@gmail.com, betmbe200@yahoo.fr, vincent.gingras.5@ulaval.ca, yanick.ouellet.2@ulaval.ca, Claude-Guy.Quimper@ift.ulaval.ca

Abstract. Fast and powerful propagators are the main key to the success of constraint programming on scheduling problems. It is, for example, the case with the CUMULATIVE constraint, which is used to model tasks sharing a resource of discrete capacity. In this paper, we propose a new not-first/not-last rule, which we call the *horizontally elastic not-first/not-last*, based on strong relaxation of the earliest completion time of a set of tasks. This computation is obtained when scheduling the tasks in a horizontally elastic way. We prove that the new rule is sound and is able to perform additional adjustments missed by the classic not-first/not-last rule. We use the new data structure called *Profile* to propose a $\mathcal{O}(n^3)$ filtering algorithm for a relaxed variant of the new rule where n is the number of tasks sharing the resource. We prove that the proposed algorithm still dominates the classic not-first/not-last algorithm. Experimental results on highly cumulative instances of resource constrained project scheduling problems (RCPSP) show that using this new algorithm can substantially improve the solving process of instances with an occasional and marginal increase of running time.

1 Introduction

Cumulative scheduling is the allocation of a scarce resource to tasks over time. It appears in many real-world problems such as university timetable, ship loading, employee scheduling, bridge or building constructions. The challenging part of this problem comes from the resource constraint. To solve it efficiently with a constraint programming solver, it is important to have fast and efficient propagators for the CUMULATIVE [1] constraint. The CUMULATIVE constraint models the problem where a limited number of tasks can be executed simultaneously.

* This work was partially supported by a grant from the Niels Henrik Abel board and the University Laval.

In a cumulative scheduling problem (CuSP), a set of tasks T has to be executed on a resource of capacity C. Each task $i \in T$ is executed without interruption during p_i time units and uses $c_i \leq C$ units of resource. For a task $i \in T$, the earliest start time est_i and the latest completion time lct_i are specified. A solution to a CuSP instance is an assignment of valid start time s_i to each task $i \in T$ such that the resource constraint is satisfied, i.e.,

$$\forall i \in T, \qquad \text{est}_i \leq s_i \leq s_i + p_i \leq \text{lct}_i \tag{1}$$

$$\forall \tau, \qquad \sum_{i \in T,\ s_i \leq \tau < s_i + p_i} c_i \leq C \tag{2}$$

The inequalities in (1) ensure that each task is assigned a feasible start and end time, while (2) enforces the resource constraint. Each task $i \in T$ has an energy $e_i = c_i \cdot p_i$, an earliest completion time $\text{ect}_i = \text{est}_i + p_i$ and a latest start time $\text{lst}_i = \text{lct}_i - p_i$. These notations can be extended to non-empty sets of tasks as follows:

$$e_\Omega = \sum_{j \in \Omega} e_j, \quad \text{est}_\Omega = \min_{j \in \Omega} \text{est}_j, \quad \text{lct}_\Omega = \max_{j \in \Omega} \text{lct}_j, \quad \text{ECT}_\Omega = \min_{j \in \Omega} \text{ect}_j. \tag{3}$$

By convention, for empty sets we have: $e_\emptyset = 0$, $\text{est}_\emptyset = +\infty$, $\text{lct}_\emptyset = -\infty$ and $\text{ECT}_\emptyset = +\infty$. Throughout the paper, we assume that for any task $i \in T$, $\text{ect}_i \leq \text{lct}_i$ and $c_i \leq C$, otherwise the problem has no solution. We let $n = |T|$ denotes the number of tasks, $k = |\{c_i, i \in T\}|$ denotes the number of distinct capacity requirements of tasks. $H = \{\text{ect}_i, i \in T\}$ denotes the set of distinct earliest completion times of tasks with $|H| = h$. The global constraint CUMULATIVE removes inconsistent values from the domain of starting time variable $s_i \in [\text{est}_i, \text{lst}_i]$. Since the CuSP is a NP-Hard problem [2], it is NP-Hard to remove all such values. Polynomial time algorithms only exist for relaxations of the problem.

The global constraint CUMULATIVE integrates many filtering algorithms which perform different pruning and are sometimes combined for more pruning (depending on the characteristics of the instance) to reduce the search space and thus the running time [3]. Each of these filtering algorithms can be called thousands of times during the search. Therefore, it is important for them to be fast, exact and efficient. Among these filtering algorithms, edge-finding [4,7] and timetabling [5] are the most used, but there exists many other filtering algorithms such as not-first/not-last [6,9,10], energetic reasoning [3,12], and more recently, horizontally elastic edge-finder [8]. For the remainder of this paper we focus solely on the algorithm for updating the earliest starting times, as the latest completion time algorithm is both symmetric and easily derived.

In this paper, we propose a new formulation of the not-first/not-last rule based on a strong relaxation of the earliest completion time of a set of tasks. The novel formula, which we call *horizontally elastic not-first/not-last* subsumes the classic not-first/not-last rule. With the data structure *Profile* from [8], we propose a $\mathcal{O}(n^3)$ relaxation of the new rule. Experimental results on highly cumulative instances of resource constrained project scheduling problems

(RCPSP) from suites benchmarks of libraries BL [11], Pack [13] and KSD15_D [14] highlight that using this new algorithm reduces the number of backtracks for a large majority of instances with an occasional and marginal increase of the running time.

The rest of the paper is organized as follows. Section 2 presents the classic not-first rule for cumulative resource constraint and Section 3 defines the novel function for computing the earliest completion time of a set of tasks with its corresponding algorithm as it is formulated and presented in [8] . In section 4, we propose a new formulation of the not-first rule based on the earliest completion time of a set of tasks which subsumes the classic not-first/not-last rule. Section 5 focuses on the presentation of a $\mathcal{O}(n^3)$ not-first algorithm for the horizontally elastic not-first rule where n is the number of tasks being scheduled on the resource, while Section 6 presents a relaxation of the above algorithm with the same complexity. Section 7 shows that the relaxed horizontally elastic not-first algorithm dominates the classic not-first algorithm. Finally, in Section 8, the empirical evaluation of the new algorithm on highly cumulative instances of the RCPSP is presented while Section 9 concludes the paper.

2 Classic not-first rule

The not-first/not-last rule detects tasks that cannot run first/last relatively to a set of tasks and prunes their time bounds. If a task i cannot be the first to be executed in $\Omega \cup \{i\}$ then the earliest start time of task i is updated to the minimum earliest completion time of the set. The not-first filtering rule is formalized as follows:
$\forall \Omega \subset T,\ \forall i \in T \setminus \Omega$

$$\begin{cases} \text{est}_i < \text{ECT}_\Omega \\ e_\Omega + c_i(\min(\text{ect}_i, \text{lct}_\Omega) - \text{est}_\Omega) > C(\text{lct}_\Omega - \text{est}_\Omega) \end{cases} \Rightarrow \text{est}_i \geq \text{ECT}_\Omega. \quad \text{(NF)}$$

Recently, in [9] the authors proposed a quadratic not-first algorithm using the Timeline data structure. Some $\mathcal{O}(n^2 \log n)$ algorithms proposed for this rule can be found in [6,10].

3 Function of the Earliest Completion Time

We present a function to compute the earliest completion time of a set of tasks as in [8]. We use the notation ect^F_Ω to denote the fully-elastic earliest completion time of a set of tasks Ω and it is computed by spending a maximum amount of energy as early as possible without any regards to the resource required of the tasks using the following formula [7].

$$\text{ect}^F_\Omega = \left\lceil \frac{\max\{C\text{est}_{\Omega'} + e_{\Omega'} | \Omega' \subseteq \Omega\}}{C} \right\rceil. \quad (4)$$

This value is a relaxation of the real earliest completion time ect_Ω of the set Ω. Note that ect_Ω is NP-hard to compute [2]. A stronger relaxation for the function ect_Ω called horizontally elastic earliest completion time and noted ect^H_Ω is introduced in [8].

During the computation of this value, any task i consumes e_i units of resource within the interval $[\text{est}_i, \text{lct}_i)$ and is allowed to consume at any time $t \in [\text{est}_i, \text{lct}_i)$, between 0 and c_i units of resource. Given a set of tasks Ω, ect^H_Ω is computed using the following functions.

- $c_{max}(t)$ the amount of resource that can be allocated to the tasks in Ω at time t, i.e.,

$$c_{max}(t) = \min\left(\sum_{i \in \Omega | \text{est}_i \leq t < \text{lct}_i} c_i, C\right) \tag{5}$$

- $c_{req}(t)$ the amount of resource required at time t by the tasks in Ω if they were all starting at their earliest starting times, i.e.,

$$c_{req}(t) = \sum_{i \in \Omega | \text{est}_i \leq t < \text{ect}_i} c_i \tag{6}$$

- $ov(t)$ called overflow is the energy from $c_{req}(t)$ that cannot be executed at time t due to the limited capacity $c_{max}(t)$.
- $c_{cons}(t)$ the amount of resource that is actually consumed at time t, i.e.,

$$c_{cons}(t) = \min(c_{req}(t) + ov(t-1), c_{max}(t)) \tag{7}$$

$$ov(t) = ov(t-1) + c_{req}(t) - c_{cons}(t) \tag{8}$$

$$ov(\text{est}_\Omega) = 0 \tag{9}$$

The horizontally elastic earliest completion time occurs when all tasks are completed, i.e.,

$$\text{ect}^H_\Omega = \max\{t | c_{cons}(t) > 0\} + 1. \tag{10}$$

For a set of tasks, it is proven in [8] that the horizontally elastic earliest completion time is a relaxation of the earliest completion time and is stronger than the fully-elastic one.

Theorem 1. *[8] For all $\Omega \subseteq T$, $ect^F_\Omega \leq ect^H_\Omega \leq ect_\Omega$.*

The computation of ect^H is done with the Profile data structure [8] that stores the resource utilization over time. The tuples $\langle time, cap, \delta_{\max}, \delta_{req} \rangle$ (where time is the start time, cap is the remaining capacity of the resource at the start time, $\delta_{\max}$ and δ_{req} are two quantities initialized to zero) are stored in a sorted linked list whose nodes are called time points. The Profile is initialized with a time point of capacity C for every distinct value of est, ect, and lct. A sufficiently large time point is added to act as a sentinel. While initializing the data structure, pointers

are kept so that t_{est_i} , t_{ect_i} and t_{lct_i} return the time point associated to est_i, ect_i, and lct_i. The algorithm *ScheduleTasks* computes the functions $c_{req}(t)$, $c_{max}(t)$, $c_{cons}(t)$ and $ov(t)$ to schedule a set of tasks Θ on the profile P.

Algorithm 1: $ScheduleTasks(\Theta, C)$ [8]

Input: Θ a set of tasks and C the capacity of the resource
Output: A lower bound ect^H_Θ of the set Θ

for *all time point* t **do**
  $t.\delta_{\max} \leftarrow 0$ and $t.\delta_{req} \leftarrow 0$
for $i \in \Theta$ **do**
  Increment $t_{\text{est}_i}.\delta_{\max}$ and $t_{\text{est}_i}.\delta_{req}$ by c_i
  Decrement $t_{\text{lct}_i}.\delta_{\max}$ and $t_{\text{ect}_i}.\delta_{req}$ by c_i
$t \leftarrow P.first$, $ov \leftarrow 0$, $\text{ect} \leftarrow -\infty$, $S \leftarrow 0$, $c_{req} \leftarrow 0$
while $t.time \neq lct_\Theta$ **do**
  $t.ov \leftarrow ov$, $l \leftarrow t.next.time - t.time$, $S \leftarrow S + t.\delta_{\max}$
  $c_{max} \leftarrow \max(S, C)$
  $c_{req} \leftarrow c_{req} + t.\delta_{req}$
  $c_{cons} \leftarrow \min(c_{req} + ov, c_{max})$
  if $0 < ov < (c_{cons} - c_{req}) \cdot l$ **then**
    $l \leftarrow \max\left(1, \left\lfloor \frac{ov}{c_{cons} - c_{req}} \right\rfloor\right)$
    $t.insertAfter(t.time + l, t.cap, 0, 0)$
  $ov \leftarrow ov + (c_{req} - c_{cons}) \cdot l$
  $t.cap \leftarrow C - c_{cons}$
  if $t.cap < C$ **then** $\text{ect} \leftarrow t.next.time$
  $t \leftarrow t.next$
return *ect*

The interesting properties of this data structure come from the number of time points and the linear-time algorithm *ScheduleTasks*.

Proposition 1 ([8]). *The Profile contains at most* $4n + 1$ *time points and the algorithm ScheduleTasks runs in* $\mathcal{O}(n)$ *time where* n *is the number of tasks.*

4 New formulation of the not-first rule

Before generalizing the not-first rule, let us state the classic not-first using the earliest completion time of a set of tasks. Let $\Omega \subset T$ be a set of tasks and $i \in T \setminus \Omega$ be a task. From task i and set of tasks Ω, a new task i' can be derived with the following attributes: $\text{est}_{i'} = \text{est}_\Omega$, $\text{lct}_{i'} = \min(\text{ect}_i, \text{lct}_\Omega)$, $p_{i'} = \min(\text{ect}_i, \text{lct}_\Omega) - \text{est}_\Omega$ and $c_{i'} = c_i$. Substituting this into rule (NF) leads to:

$$\begin{cases} \text{est}_i < \text{ECT}_\Omega \\ e_\Omega + e_{i'} > C(\text{lct}_\Omega - \text{est}_{\Omega \cup \{i'\}}) \end{cases} \Rightarrow \text{est}_i \geq \text{ECT}_\Omega \tag{11}$$

The rule (11) is equivalent to

$$\begin{cases} \text{est}_i < \text{ECT}_\Omega \\ \text{ect}^F_{\Omega \cup \{i'\}} > \text{lct}_\Omega \end{cases} \Rightarrow \text{est}_i \geq \text{ECT}_\Omega \tag{12}$$

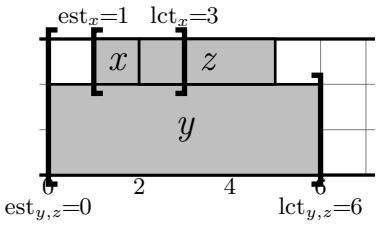

Fig. 1. (a) A CuSP problem of 3 tasks sharing a resource of capacity $C = 3$.

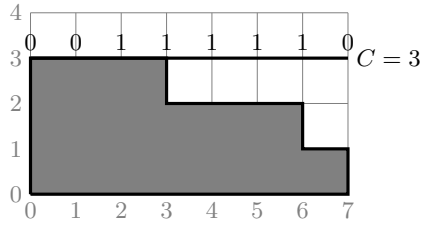

Fig. 2. (b) The resource utilization profile of tasks $\{x, y, z'\}$ where z' is derived from z and $\Omega = \{x, y\}$ and whose parameters $\langle \text{est}_{z'}, \text{lct}_{z'}, p_{z'}, c_{z'} \rangle$ are $\langle 0, 3, 3, 1 \rangle$.

The horizontally elastic not-first rule is obtained from (12) by replacing $\text{ect}^F_{\Omega \cup \{i'\}}$ by $\text{ect}^H_{\Omega \cup \{i'\}}$ and is given by the formula:
$\forall \Omega \subset T,\ \forall i \in T \setminus \Omega$,

$$\begin{cases} \text{est}_i < \text{ECT}_\Omega \\ \text{ect}^H_{\Omega \cup \{i'\}} > \text{lct}_\Omega \end{cases} \Rightarrow \text{est}_i \geq \text{ECT}_\Omega \qquad \text{(HNF)}$$

where i' is a task derived from task i whose parameters $\langle \text{est}_{i'}, \text{lct}_{i'}, p_{i'}, c_{i'} \rangle$ are $\langle \text{est}_\Omega, \min(\text{ect}_i, \text{lct}_\Omega), \min(\text{ect}_i, \text{lct}_\Omega) - \text{est}_\Omega, c_i \rangle$.

Theorem 2. *The not-first rule* (NF) *is subsumed by the horizontally elastic not-first rule* (HNF).

Proof. Since $\text{ect}^F_\Omega \leq \text{ect}^H_\Omega$ for all Ω (Theorem 1) and from the equivalence of rules (NF) and (12), it follows that condition $e_\Omega + c_i(\min(\text{ect}_i, \text{lct}_\Omega) - \text{est}_\Omega) > C(\text{lct}_\Omega - \text{est}_\Omega)$ implies the condition $\text{ect}^H_{\Omega \cup \{i'\}} > \text{lct}_\Omega$. Therefore, all the adjustments performed by rule (NF) are also done by rule (HNF).

Consider the CuSP instance of Figure 1 where three tasks share a resource of capacity $C = 3$. The resource utilization profile of tasks $\{x, y, z'\}$ is given in Figure 2 where z' is derived from z and $\Omega = \{x, y\}$ whose parameters $\langle \text{est}_{z'}, \text{lct}_{z'}, p_{z'}, c_{z'} \rangle$ are $\langle 0, 3, 3, 1 \rangle$. The numbers above the bold line of capacity limit are overflow units of energy remaining at each time point i.e., $ov(0) = 0$, $ov(1) = 0$, $ov(2) = 1$, and so forth. The horizontally elastic earliest completion time of the tasks set $\{x, y, z'\}$ is 7 which is greater than $\text{lct}_{\{x,y,z'\}} = 6$. When we apply the not-first rule (NF) with $\Omega = \{x, y\}$ and $i = z$, it appears that $0 = \text{est}_z < 2 = \text{ECT}_\Omega$ but $\text{ect}^F_{\Omega \cup \{z'\}} = 3 \leq \text{lct}_\Omega$. Therefore, no detection is found and consequently no adjustment follows. But the horizontally elastic not-first rule (HNF) applied to the same instance gives $\text{ect}^H_{\Omega \cup \{z'\}} = 7 > \text{lct}_\Omega$ and the earliest start time of task z is updated to 2.□

5 Horizontally Elastic Not-First Algorithm

We present a $\mathcal{O}(n^3)$ cumulative horizontally elastic not-first algorithm where n is the number of tasks sharing the resource. The new algorithm is sound as in

[6,10] i.e., the algorithm may take additional iterations to perform maximum adjustments and uses the concept of the left cut of the set of tasks T by a task j as in [6]. We describe how the left cut can be used to check the not-first conditions. We present some strategies to reduce the practical complexity of the algorithm and to fully utilize the power of the the Profile data structure.

Definition 1. *[6] Let i and j be two different tasks. The left cut of T by task j relatively to task i is the set of tasks $LCut(T, j, i)$ defined as follows:*

$$LCut(T, j, i) = \{k \mid k \in T \wedge k \neq i \wedge est_i < ect_k \wedge lct_k \leq lct_j\}. \tag{13}$$

Using this set, we have the following new rule:

For all $i, j \in T$ with $i \neq j$,

$$\mathrm{ect}^H_{\mathrm{LCut}(T,j,i)\cup\{i'\}} > \mathrm{lct}_j \Rightarrow \mathrm{est}_i \geq \mathrm{ECT}_{\mathrm{LCut}(T,j,i)} \tag{HNF'}$$

where i' is a task derived from task i whose parameters $\langle \mathrm{est}_{i'}, \mathrm{lct}_{i'}, p_{i'}, c_{i'}\rangle$ are $\langle \mathrm{est}_{\mathrm{LCut}(T,j,i)}, \min(\mathrm{ect}_i, \mathrm{lct}_j), \min(\mathrm{ect}_i, \mathrm{lct}_j) - \mathrm{est}_{\mathrm{LCut}(T,j,i)}, c_i\rangle$.

Theorem 3. *For a task i, at most $h-1$ iterative applications of the rule* (HNF') *achieve the same filtering as one application of the rule* (HNF) *where h is the number of different earliest completion time of tasks.*

Proof. Let Ω be the set which induces the maximum change of the value est_i by the rule (HNF). Let $j \in \Omega$ be a task with $\mathrm{lct}_j = \mathrm{lct}_\Omega$. Until the same value of est_i is reached, in each iteration of the rule (HNF') holds that $\Omega \subseteq \mathrm{LCut}(T, j, i)$. Indeed, because $\mathrm{est}_i < \mathrm{ECT}_\Omega$ and $i \notin \Omega$, it follows that for all $k \in \Omega$, $k \neq i$, $\mathrm{est}_i < \mathrm{ect}_k$ and $\mathrm{lct}_k \leq \mathrm{lct}_j$. From the inclusion $\Omega \subseteq \mathrm{LCut}(T, j, i)$, it follows that $\mathrm{ect}^H_{\mathrm{LCut}(T,j,i)\cup\{i'\}} \geq \mathrm{ect}^H_{\Omega\cup\{i'\}} > \mathrm{lct}_j$ and the rule (HNF') holds and propagates.

After each successful application of the rule (HNF'), the value est_i is increased. This removes all the tasks from the set $\mathrm{LCut}(T, j, i)$ having the earliest completion time $\mathrm{ECT}_{\mathrm{LCut}(T,j,i)}$. Therefore the final value of est_i must be reached after at most $h - 1$ iterations and it is the same as for the rule (HNF).□

Example 1 ([6]). Consider the CuSP instance of Figure 3 where four tasks share

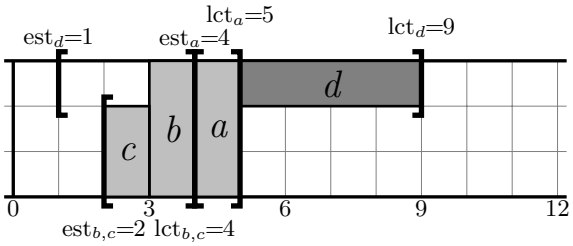

Fig. 3. A scheduling problem of 4 tasks sharing a resource of capacity $C = 3$.

a resource of capacity $C = 3$. The not-first rule (HNF') holds for task d and the set $\text{LCut}(T, a, d) = \{a, b, c\}$. Indeed, $\text{est}_d < \text{ECT}_{\{a,b,c\}}$ and $\text{ect}^H_{\text{LCut}(T,a,d)\cup\{d'\}} = 6 > \text{lct}_a$. Hence, $\text{est}_d = 3$. After this adjustment, we have $\text{LCut}(T, a, d) = \{a\}$, $\text{est}_d < \text{ECT}_{\{a\}}$, and $\text{ect}^H_{\text{LCut}(T,a,d)\cup\{d'\}} = 6 > \text{lct}_a$ which leads to $\text{est}_d = 5$. The maximum adjustment holds.

To reduce the practical computational complexity of the algorithm, we deduce from the properties of $\text{LCut}(T, j, i)$ and the rigidity of task i' some strategies to learn from failures and successes and anticipate the detection of future tasks.

5.1 Reducing the number of sets $\Theta = \text{LCut}(T, j, i)$ to consider

To speed-up the algorithm without reducing its filtering power, we can know whether or not the set of tasks $\Theta = \text{LCut}(T, j, i)$ is in conflict with the task i. The detection of the not-first rule of task i with the set of tasks Θ is only possible when Θ is conflicting with i. This happens when $\sum_{k\in\Theta} c_k > C - c_i$ since when the sum of the capacity requirements of the tasks in Θ is less than $C - c_i$, then the set Θ is not conflicting with task i.

5.2 Deduction From Detection Failure of Tasks

Let i be a task such that the not-first detection rule (HNF') fails for all set of tasks $\text{LCut}(T, j, i)$. Then for any other task u such that $u \neq i$, $\text{lct}_u \leq \text{lct}_i$, $\text{est}_u = \text{est}_i$, $c_u \leq c_i$ and $\text{ect}_u \leq \text{ect}_i$, we can deduce that, for all sets of tasks $\text{LCut}(T, j, i)$ the rule (HNF') will also fail with task u. This assertion is formally proven in the following proposition.

Proposition 2. *Let $i \in T$ be a task such that the not-first rule* (HNF') *fails for all sets of tasks $LCut(T, j, i)$. Then for any task $u \in T$ such that $u \neq i$, $lct_u \leq lct_i$, $est_u = est_i$, $c_u \leq c_i$ and $ect_u \leq ect_i$, the not-first rule* (HNF') *also fails with task u for all sets of tasks $LCut(T, j, u)$.*

Proof. By contradiction, let $u \in T$ be a task with $u \neq i$, $\text{lct}_u \leq \text{lct}_i$, $\text{est}_u = \text{est}_i$, $c_u \leq c_i$ and $\text{ect}_u \leq \text{ect}_i$ such that the not-first rule (HNF') detects and adjusts the earliest start time of task u, i.e., there exists a task $j \in T$ such that $\text{ect}^H_{\text{LCut}(T,j,u)\cup\{u'\}} > \text{lct}_j$ and the task u is updated such that $\text{est}_u \geq \text{ECT}_{\text{LCut}(T,j,u)}$. We distinguish two cases : $\text{lct}_j < \text{lct}_i$ and $\text{lct}_i \leq \text{lct}_j$.

1. If $\text{lct}_j < \text{lct}_i$, then $\text{LCut}(T, j, u) \subseteq \text{LCut}(T, j, i)$ and from the fact that i' is more constrained than u' it follows that $\text{ect}^H_{\text{LCut}(T,j,i)\cup\{i'\}} \geq \text{ect}^H_{\text{LCut}(T,j,i)\cup\{u'\}} \geq \text{ect}^H_{\text{LCut}(T,j,u)\cup\{u'\}} > \text{lct}_j$, which contradicts the non-detection of the not-first rule (HNF') with task i.
2. If $\text{lct}_i \leq \text{lct}_j$, then $\text{LCut}(T, j, u) \subseteq \text{LCut}(T, j, i) \cup \{i\}$. Since the set of tasks $\text{LCut}(T, j, i) \cup \{i'\}$ is more constrained than $\text{LCut}(T, j, u) \cup \{u'\}$ it follows that $\text{ect}^H_{\text{LCut}(T,j,i)\cup\{i'\}} \geq \text{ect}^H_{\text{LCut}(T,j,u)\cup\{u'\}} > \text{lct}_j$, which contradicts the non-detection of the rule (HNF') with task i. □

5.3 Deduction From Success Detection of Tasks

Let i and j be two tasks such that $i \neq j$, $\text{est}_i < \text{ect}_j$ and the not-first rule (HNF') holds for i and $\text{LCut}(T, j, i)$. Then for any other task u such that $u \neq i$, $\text{lct}_u \leq \text{lct}_i$, $\text{est}_u \leq \text{est}_i$, $c_u \geq c_i$ and $\text{ect}_u \geq \text{ect}_i$, the tasks set $\text{LCut}(T, j, u)$ successfully detected the not-first rule (HNF') with task u, $u \notin \text{LCut}(T, j, i)$. This assertion is formally proven in the following proposition.

Proposition 3. *Let i and j be two tasks such that $i \neq j$, $est_i < ect_j$ and the not-first rule* (HNF') *holds for i and $LCut(T, j, i)$. Then for any other task u such that $u \neq i$, $lct_u \leq lct_i$, $est_u \leq est_i$, $c_u \geq c_i$ and $ect_u \geq ect_i$, if $u \notin LCut(T, j, i)$ then the not-first rule* (HNF') *holds with u and $LCut(T, j, u)$.*

Proof. Let i and j be two tasks such that $i \neq j$, $\text{est}_i < \text{ect}_j$ and the rule (HNF') holds for i and $\text{LCut}(T, j, i)$. Let u be a task such that $u \neq i$, $\text{lct}_u \leq \text{lct}_i$, $\text{est}_u \leq \text{est}_i$, $c_u \geq c_i$, $\text{ect}_u \geq \text{ect}_i$ and $u \notin \text{LCut}(T, j, i)$. From $\text{est}_u \leq \text{est}_i$ and $u \notin \text{LCut}(T, j, i)$, it is obvious that $\text{LCut}(T, j, i) \subseteq \text{LCut}(T, j, u)$. Since the set of tasks $\text{LCut}(T, j, u) \cup \{u'\}$ is more constrained than $\text{LCut}(T, j, i) \cup \{i'\}$ it follows that $\text{ect}^H_{\text{LCut}(T,j,u)\cup\{u'\}} \geq \text{ect}^H_{\text{LCut}(T,j,i)\cup\{i'\}} > \text{lct}_j$. □

To apply these reductions, we start with a set $\Lambda = T$ of tasks sorted by non-decreasing order of lct_j, by non-increasing order of est_j, by non-decreasing order of c_j and by non-decreasing order of ect_j to break ties. If a task $i \in \Lambda$ fails for detection of the rule (HNF'), then we are sure that the detection will fail with all tasks $u \in \Lambda$ such that $u \neq i$, $\text{lct}_u \leq \text{lct}_i$, $\text{est}_u = \text{est}_i$, $c_u \leq c_i$ and $\text{ect}_u \leq \text{ect}_i$. On the other hand, when the rule (HNF') holds with a task i and the set $\text{LCut}(T, j, i)$, if the detection of the rule (HNF') fails with the set $\text{LCut}(T, j, u)$ and the task $u \in \Lambda$ such that $u \neq i$, $\text{lct}_u \leq \text{lct}_i$, $\text{est}_u \leq \text{est}_i$, $c_u \geq c_i$ and $\text{ect}_u \geq \text{ect}_i$, then the task is postponed to the next iteration.

5.4 Horizontally elastic Not-First Algorithm

In Algorithm 2, we iterate through the set of tasks sorted by non-decreasing order of lct, by non-increasing order of est, by non-decreasing order of c_j and by non-decreasing order of ect_j to break ties (line 5) and for each unscheduled task (line 3), we iterate over the different set $\Theta = \text{LCut}(T, j, i)$ (line 7). For each set Θ satisfying the reduction of section 5.1, the minimum earliest completion time is computed (line 9) during the initialization of the increment values of the function *ScheduleTasks*. The horizontally-elastic earliest completion time of the set of tasks $\Theta \cup \{i'\}$ is computed at line 10 by the function *ScheduleTasks* and if this value is greater than lct_j (line 11), then the adjustment of est_i occurs (line 12). The boolean "detect" of line 4 allows breaking for the while loop of line 5 when a detection is found. The loop of line 15 is used to avoid similar set $\Theta = \text{LCut}(T, j, i)$ since $\text{LCut}(T, j, i) = \text{LCut}(T, j', i)$ if $\text{lct}_j = \text{lct}_{j'}$. The complexity of Algorithm 2 is given in the following theorem.

Algorithm 2: Horizontally elastic Not-First Algorithm in $\mathcal{O}(n^3)$ time.

Input: Λ tasks sorted by non-decreasing lct_j, by non-increasing est_j, by non-decreasing c_j and by non-decreasing ect_j to break ties.
Output: A lower bound est'_i for each task i

```
1  for i ∈ T do est'_i ← est_i
2  for i = n to 1 do
3      if ect_i < lct_i then
4          detect ← false, j ← 1, t ← −1
5          while j ≤ n ∧ detect = false do
6              if j ≠ i ∧ est_i < ect_j then
7                  Θ ← LCut(T, j, i)
8                  if Σ_{k∈Θ} c_k > C − c_i then
9                      ECT ← ECT_{LCut(T,j,i)}
10                     ect^H ← ScheduleTasks(Θ ∪ {i'}, C)
11                     if ect^H > lct_j then
12                         est'_i ← max(est'_i, ECT)
13                         detect ← true
14                         t ← j
15                     while j + 1 ≤ n ∧ lct_j = lct_{j+1} ∧ detect = false do
16                         j ← j + 1
17             j ← j + 1
18         if detect = true then
19             for u = i − 1 to 1 do
20                 if est_u ≤ est_i ∧ c_u ≥ c_i ∧ ect_u ≥ ect_i then
21                     Θ ← LCut(T, t, u)
22                     ECT ← ECT_{LCut(T,t,u)}
23                     ect^H ← ScheduleTasks(Θ ∪ {u'}, C)
24                     if ect^H > lct_j then
25                         est'_u ← max(est'_u, ECT)
26                     Λ ← Λ \ {u}
27         if detect = false then
28             for u = i − 1 to 1 do
29                 if est_u = est_i ∧ c_u ≤ c_i ∧ ect_u ≤ ect_i then
30                     Λ ← Λ \ {u}
31 for i ∈ T do est_i ← est'_i
```

Theorem 4. *Algorithm 2 runs in $\mathcal{O}(n^3)$ time.*

Proof. The linear time algorithm *ScheduleTasks* is called $\mathcal{O}(n^2)$ time for total complexity of $\mathcal{O}(n^3)$.□

We perform a preliminary comparison of this algorithm with the state of the art algorithms on the resource constrained project scheduling problems (RCPSP)

instances of the BL set [3]. It appears that on many instances, when the proposed algorithm is used, the solver spends 1.5 - 2 more time than with the others not-first/not-last algorithms for a reduction of the number of backtracks of less than 40%. We observe that for a task i, many set $\Theta = \text{LCut}(T, j, i)$ used to check the not-first conditions are fruitless and should be avoided.

6 Relaxation of the Horizontally Elastic Not-First Algorithm

We propose a relaxation of the previous algorithm based on a new criterion used to reduce the number of subsets $\text{LCut}(T, j, i)$ to consider. Without changing the computational complexity, the relaxed horizontally elastic not-first algorithm still dominates the classic not-first algorithm with a good trade-off between the filtering power and the running time. To do so, it is important to have a criteria to select the task j for which the set $\Theta = \text{LCut}(T, j, i)$ has more potential to detect at least the classic not-first conditions.

Definition 2. *Let $i \in T$ be a task. The not-first set of tasks with task i denoted $NFSet(T, i)$ is given by*

$$NFSet(T, i) = \{j, j \in T \wedge j \neq i \wedge est_i < ect_j\}.$$

The set $\text{NFSet}(T, i)$ is the set of tasks conflicting with task i. If a not-first condition is detected with a set Ω i.e., $\text{ect}_{\Omega \cup \{i'\}} > \text{lct}_\Omega$, then $\Omega \subseteq \text{NFSet}(T, i)$. In this condition, the earliest start time of task i' can be replaced by $\text{est}_{\min} = \min\{\text{est}_k, k \in T\}$ since none of the tasks from $\text{NFSet}(T, i)$ starts and ends before est_i. We schedule the tasks from $\text{NFSet}(T, i) \cup \{i'\}$ and compute the overflow energy that cannot be executed at time $t = t_{\text{lct}_j}$ for $j \in \text{NFSet}(T, i)$. The algorithm *ScheduleNFConflictingTasks* is a variant of the algorithm *ScheduleTasks* which schedules the set $\text{NFSet}(T, i) \cup \{i'\}$ and returns the set Δ of task j such that the overflow energy at time point t_{lct_j} is greater than 0.

We use the condition $t_{\text{lct}_j}.ov > 0$ to reduce the number of sets $\text{LCut}(T, j, i)$ to be considered during the detection of the not-first conditions with task i. The above improvements are incorporated in Algorithm 3. The complexity of the resulting algorithm remains $\mathcal{O}(n^3)$ but the condition $t_{\text{lct}_j}.ov > 0$ used to reduce the number of sets $\text{LCut}(T, j, i)$ during the detection considerably reduces the running time, as shown from the experimental results section.

Example 2. Consider the CuSP instance of Figure 1 with an additional task t where attributes $\langle \text{est}_t, \text{lct}_t, p_t, c_t \rangle$ are $\langle 3, 7, 1, 1 \rangle$. The function *ScheduleNFConflictingTasks* return an empty set when the set $\text{NFSet}(T, z) \cup \{z'\}$ is scheduled because the overflow energy will be consumed before the time point 6. Therefore, the relaxed algorithm will miss the adjustment of est_z to 2.

The filtering power of the algorithm is reduced. We prove later that the relaxed horizontally elastic not-first algorithm subsumes the classic not-first algorithm.

Algorithm 3: Relaxation of the horizontally elastic Not-First in $\mathcal{O}(n^3)$.

Input: Λ tasks sorted by non-decreasing lct_j, by non-increasing est_j, by non-decreasing c_j and by non-decreasing ect_j to break ties.
Output: A lower bound est'_i for each task i

```
1  for i ∈ T do est'_i ← est_i
2  for i = n to 1 do
3      if ect_i < lct_i ∧ i ∈ Λ then
4          detect ← false, t ← −1
5          if detect = false then
6              Δ ← ScheduleNFConflictingTasks(i, C)
7              j ← |Δ|
8              while j ≥ 1 ∧ detect = false do
9                  Θ ← LCut(T, j, i)
10                 ECT ← ECT_{LCut(T,j,i)}
11                 ect^H ← ScheduleTasks(Θ ∪ {i'}, C)
12                 if ect^H > lct_j then
13                     est'_i ← max(est'_i, ECT)
14                     detect ← true
15                     t ← j
16                 j ← j − 1
17         if detect = true then
18             for u = i − 1 to 1 do
19                 if est_u ≤ est_i ∧ c_u ≥ c_i ∧ ect_u ≥ ect_i then
20                     Θ ← LCut(T, t, u)
21                     ECT ← ECT_{LCut(T,t,u)}
22                     ect^H ← ScheduleTasks(Θ ∪ {u'}, C)
23                     if ect^H > lct_t then
24                         est'_u ← max(est'_u, ECT)
25                     Λ ← Λ \ {u}
26         if detect = false then
27             for u = i − 1 to 1 do
28                 if est_u = est_i ∧ c_u ≤ c_i ∧ ect_u ≤ ect_i then
29                     Λ ← Λ \ {u}
30 for i ∈ T do est_i ← est'_i
```

7 Properties of the relaxation of the horizontally elastic not-first algorithm

In this section, we prove that the relaxation of the horizontally elastic not-first algorithm (Algorithm 3) subsumes the standard not-first algorithm.

Lemma 1. *Let $i \in T$ be a task. If the not-first condition* (NF) *is detected with the set of tasks Ω, then after a horizontally elastic scheduling of tasks $NFSet(T, i) \cup \{i'\}$, it appears that $t_{lct_j}.ov > 0$ where $lct_j = lct_\Omega$.*

Proof. Let $j \in T$ be a task such that $\text{lct}_j = \text{lct}_\Omega$. If the not-first conditions (NF) are detected with the set of tasks Ω, then $\text{ect}^F_{\Omega \cup \{i'\}} > \text{lct}_j$ and $\Omega \subseteq \text{LCut}(T, j, i)$. Therefore, in the fully elastic schedule of tasks from $\Omega \cup \{i'\}$, the resource is fully used at any time points from est_Ω to lct_Ω with a surplus of energy not executed. Then from $\text{ect}^H_{\text{LCut}(T,j,i)\cup\{i'\}} \geq \text{ect}^F_{\text{LCut}(T,j,i)\cup\{i'\}} \geq \text{ect}^F_{\Omega\cup\{i'\}} > \text{lct}_j$ and $\text{LCut}(T, j, i) \subseteq \text{NFSet}(T, i)$ it follows that during the scheduling of tasks set $\text{NFSet}(T, i) \cup \{i'\}$, $t_{\text{lct}_j}.ov > 0$.□

Theorem 5. *The relaxation of the horizontally elastic not-first algorithm (Algorithm 3) subsumes the classic not-first algorithm.*

Proof. According to Lemma 1, any detection and adjustment performed by the classic not-first algorithm are also detected and adjusted by the relaxed horizontally elastic not-first algorithm. In the CuSP instance of Example 1, the classic not-first algorithm fails to adjust est_z while the relaxation of the horizontally elastic not-first algorithm succeeds to update est_z to 2.□

We know from [3] that the classic not-first/not-last rule is not subsumed by the energetic reasoning rule and vice-versa. According to Theorem 5, we can deduce that the relaxation of the horizontally elastic not-first/not-last rule is not subsumed by the energetic reasoning and vice-versa.

8 Experimental Results

We carry out experimentations on resource-constrained project scheduling problems (RCPSP) to compare the new algorithm of not-first/not-last with the state-of-the art algorithms. A RCPSP consists of a set of resources of finite capacities, a set of tasks of given processing times, an acyclic network of precedence constraints between tasks, and a horizon (a deadline for all tasks). Each task requires a fixed amount of each resource over its execution time. The problem is to find a starting time assignment for all tasks satisfying the precedence and resource capacity constraints, with the least makespan (i.e., the time at which all tasks are completed) at most equals to the horizon.

Tests were performed on benchmark suites of RCPSP known to be highly cumulative [3]. On highly cumulative scheduling instances, many tasks can be scheduled simultaneously as contrary to the highly disjunctive ones. We use the libraries BL [11], Pack [13] and KSD15_D [14]. The data set BL consists of 40 instances of 20 and 25 tasks sharing three resources, Pack consists of 55 instances of 15-33 tasks sharing a resource of capacity 2-5 while the set KSD15_D consists of 480 instances of 15 tasks sharing a resource of capacity 4.

Starting with the provided horizon as an upper bound, we modeled each problem as an instance of Constraint Satisfaction Problem (CSP); variables are start times of tasks and they are constrained by the precedence graph (i.e., precedence relations between pairs of tasks were enforced with linear constraints) and resource limitations (i.e., each resource was modeled with a single CUMULATIVE constraint [1]). We used a branch and bound search to minimize the makespan.

We implemented three different propagators of the global constraint CUMULATIVE in Java using Choco solver 4.0.1 [17].

1. The first CUMULATIVE propagator noted "TT-NF" (for not-first with Θ -tree) is a sequence of two filtering algorithms: the $\mathcal{O}(n^2 \log n)$ not-first algorithm from [6] and timetabling algorithm from [15].
2. The second propagator noted "CHE-NF" (for not-first with complete horizontally elastic) is obtained when replacing in the first propagator the not-first algorithm with Timeline by the complete horizontally elastic not-first algorithm presented in Algorithm 2.
3. The third propagator noted "RHE-NF" (for not-first with relaxed horizontally elastic) is obtained when replacing in the first propagator the not-first algorithm with Timeline by the relaxed horizontally elastic not-first algorithm presented in Algorithm 3.

Branching scheme is another ingredient to accelerate the solving process. The heuristics used to select tasks and values are directly linked to the type of problems and the filtering algorithms considered in the solver. We combine the conflict-ordering search heuristic [16] with the heuristic $minDomLBSearch$ from Choco. During the search, the solver records conflicting tasks and at the backtrack, the last one is selected in priority until they are all instantiated without causing any failure. When no conflicting tasks is recorded, the heuristic $minDomLBSearch$ which consists of selecting the unscheduled tasks with the smallest domain and assigning it to its lower bound is used. Tests were performed on a data center equiped with Intel(R) Xeon X5560 Nehalem nodes, 2 CPUs per node, 4 cores per CPU at 2.4 GHz, 24 GB of RAM per node. Any search taking more than 10 minutes was counted as a failure.

In Table 1, the columns "solve" report the number of instances solved by each propagator. Columns "time", "backt", and "speedup" denote the average CPU time (in second) used to reach the optimal solution, the average number of backtracks, and the average speedup factor (TT-NF time over new algorithms time) reported on instances solved by "TT-NF" vs. "CHE-NF" (sp_1) and "TT-NF" vs. "RHE-NF" (sp_2) respectively. 527 instances were solved by the three propagators with one instance solved only by "CHE-NF" and "RHE-NF" (pack016) and two instances solved only by "TT-NF" and "RHE-NF" (pack015 and j30_45_2).

	TT-NF			CHE-NF			RHE-NF			Speedup (%)	
	solve	time	backts	solve	time	backts	solve	time	backts	sp_1	sp_2
BL	40	**4.497**	32789	40	6.952	**23114**	40	6.616	27193	64.7	68
Pack	**19**	30.524	161467	18	42.608	90663	18	**24.543**	**66154**	71.6	**124.4**
KSD15_D	**471**	0.766	2196	470	1.2	**2008**	**471**	**0.743**	2020	63.8	**103.1**

Table 1. We report the number of instances solved (solve), the average number of backtracks (backts), the average time in second (time) and the average speedup factor (TT-NF time over new algorithms time) required to solve all instances that are commonly solved by the three propagators on set BL, Pack and KSD15_D.

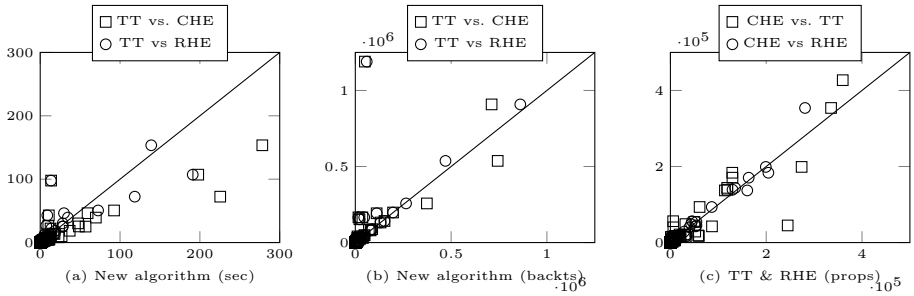

Fig. 4. (a) Runtimes comparison of TT-NF vs. CHE-NF and TT-NF vs. RHE-NF, (b) Comparison of the number of Backtracts TT-NF vs. CHE-NF and TT-NF vs. RHE-NF, (c) Comparison of the number of adjustments (Propagations) CHE-NF vs. TT-NF and CHE-NF vs. RHE-NF on instances of BL, Pack and KSD15_D where the three propagators found the best solution.

The propagator "TT-NF" performs better in average on BL set while "RHE-NF" is the best on Pack and KSD15_D with an average speedup factor of 124.4% and 103.1% wrt. "TT-NF". We observe a reduction of the average number of backtracks from "RHE-NF" on Pack set while "CHE-NF" dominated on BL and KSD15_D. Figure 4 compares the runtimes (a), the number of backtracks (b) and the number of adjustments (propagations) (c) made at the fixed point of the node of the search tree on the 527 instances solved by the three propagators. It appears in (a) that the running time of "RHE-NF" is generally close to "TT-NF" and sometimes less. In (b), the number of backtracks of "RHE-NF" is always less than the number of backtracks of "TT-NF". In (c), the average number of propagations of "RHE-NF" are always less than the average number of propagations of "CHE-NF" when on a few number of instances, the number of propagations of "TT-NF" is less than the number of propagations of "CHE-NF".

9 Conclusion

We proposed a generalization of the not-first/not-last rule for the cumulative resource constraint based on a strong relaxation of the earliest completion time of a set of tasks. A relaxation of the corresponding horizontally elastic not-first/not-last algorithm running in $\mathcal{O}(n^3)$ is also proposed, where n is the number of tasks sharing the resource. The new algorithm is sound and can reach a better fixed point than the state-of-the-art algorithms. The new algorithm is based on the data structure Profile used to compute a strong lower bound on the earliest completion time of a set of tasks. Experimental results demonstrate that the new algorithm has more impact in terms of backtracks reduction and running time on highly cumulative instances of RCPSPs. Future work will focus on finding how to improve the complexity of this algorithm from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2 \log n)$ and to design a branching scheme more suitable for the new rule.

References

1. A. Aggoun, and N. Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathematical and Computer Modeling*, 17(7), 57–73, (1993).
2. M. R. Garey and D. S. Johnson. Computers and Intractability. Volume 29, wh freeman. (2002).
3. P. Baptiste, C. Le Pape, and W. Nuijten: Constraint-based scheduling: applying constraint programming to scheduling problems. Kluwer, Boston (2001).
4. R. Kameugne, L. P. Fotso, J. Scott, Y. Ngo-Kateu: A Quadratic Edge-Finding Filtering Algorithm for Cumulative Resource Constraints, Vol. **19**. No. 3, pp 243-269. Springer, (2014).
5. S. Gay, R. Hartert, and P. Schaus. Simple and Scalable Time-Table Filtering for the Cumulative Constraint. In Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP 2015), 149–157, 2015.
6. R. Kameugne, and L.P. Fotso. A Cumulative Not-First/Not-Last Filtering Algorithm in $\mathcal{O}(n^2 \log(n))$. Indian Journal of Pure Applied Mathematics. Vol. 44 No. 1, 95–115, 2013,
7. P. Vilìm. Edge Finding Filtering Algorithm for Discrete Cumulative Resources in $\mathcal{O}(kn \log n)$. In Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP 2009). 802–816, 2009.
8. V. Gingras and C.-G. Quimper. Generalizing the Edge-Finder Rule for the Cumulative Constraint. In Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016), 3103–3109, 2016.
9. H. Fahimi, Y. Ouellet, and C.-G. Quimper. Faster algorithms for the constraints Disjunctive and Cumulative using the time line data structure. (under review) 2017.
10. A. Schutt and A. Wolf: A New $\mathcal{O}(n^2 \log n)$ Not-First/Not-Last Pruning Algorithm for Cumulative Resource Constraints. In Proceedings of the 16th International Conference on Principles and Practice of Constraint Programming (CP 2010). 445–459, 2010.
11. P. Baptiste and C. Le Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. Constraints Vol 5, No 1-2, 119-139, 2000.
12. A. Derrien and T. Petit. A new characterization of relevant intervals for energetic reasoning. Principles and Practice of Constraint Programming (CP 2014), Lecture Notes in Computer Science, 8656, pp. 289297, 2014.
13. J. Carlier and E. Néron. On linear lower bounds for the resource constrained project scheduling problem. European Journal of Operational Research, 149(2):314-324, 2003.
14. O. Koné, C. Artigues, P. Lopez, and M. Mongeau.: Event-based milp models for resource-constrained project scheduling problems. Computers & Operations Research, 38(1):3-13, 2011.
15. A. Letort, N. Beldiceanu, and M. Carlsson. A scalable sweep algorithm for cumulative constraint. In Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP 2012), 439-454, 2012.
16. Steven Gay, Renaud Hartert, Christophe Lecoutre and Pierre Schaus. Conflict Ordering Search for Scheduling Problems. n Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP 2015), 140-148, Cork, Ireland, 2015.

17. C. Prud'homme, J.-G. Fages, and X. Lorca. Choco Solver Documentation, TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016, http://www.choco-solver.org