A CUMULATIVE NOT-FIRST/NOT-LAST FILTERING ALGORITHM IN $O(n^2 \log(n))$\textsuperscript{1}

Roger Kameugne*,** and Laure Pauline Fotso***

*University of Maroua, Higher Teachers’ Training College, Dept. of Mathematics, P.O. Box 55 Maroua-Cameroon
**University of Yaoundé I, Faculty of Sciences, Dept. of Mathematics, PO Box 812, Yaoundé, Cameroon
***University of Yaoundé I, Faculty of Sciences, Dept. of Computer Sciences, PO Box 812, Yaoundé, Cameroon

*e-mails: rkameugne@yahoo.fr, rkameugne@gmail.com; lpfotso@ballstate.bsu.edu

(Received 30 July 2011; after final revision 29 November 2011; accepted 6 December 2012)

In cumulative and disjunctive constraint-based scheduling, the resource constraint is enforced by several filtering rules. Among these rules, we have (extended) edge-finding and not-first/not-last rules. The not-first/not-last rule detects tasks that cannot run first/last relatively to a set of tasks and prunes their time bounds. In this paper, it is presented a sound $O(n^2 \log n)$ algorithm for the cumulative not-first/not-last rule where $n$ is the number of tasks. This algorithm reaches the same fix point as previous not-first/not-last algorithms, although it may take additional iterations to do so. The worst case complexity of this new algorithm for the maximal adjustment is the same as

\textsuperscript{1}A preliminary version of this paper was accepted to CPDP 2010 [5].
our previous complete $O(n^2|H|\log n)$ not-first/not-last algorithm [7] where $|H|$ is the maximum between the number of distinct earliest completion and latest start times of tasks. But, experimental results on benchmarks from the Project Scheduling Problem Library (PSPLib) and the Baptiste and Le Pape data set (BL) suggest that the new not-first/not-last algorithm has a substantially reduced runtime. Furthermore, the results demonstrate that in practice the new algorithm rarely requires more propagations than previous not-first/not-last algorithms.

**Key words**: Constraint-based scheduling; global constraints; not-first/not-last; cumulative resources; cumulative Θ-tree; energy envelope.

1. Introduction

The cumulative scheduling constraint, which enforces the sharing of a finite resource by several tasks, is widely used in constraint-based scheduling applications. Propagation of the cumulative constraint can be performed by several different filtering algorithms, often used in combination (overload checking [16, 19], (extended) edge-finding [8, 11, 12, 17, 18] and not-first/not-last [5-7, 13, 14]). As (extended) edge-finding, the not-first/not-last rule is a constraint propagation technique for scheduling problems which determines the position of a single task $i$ relatively to a set of tasks $\Omega$ all sharing the same resource. More precisely, the rule checks whether the task $i$ cannot run first/last relatively to a set of tasks $\Omega$ and updates the release/deadline of task $i$ accordingly. The not-first/not-last filtering rule is well understood for disjunctive scheduling problems, i.e., problems with unary resources (resource of capacity $C = 1$). Indeed, there exist efficient algorithms running in $O(n \log n)$ time where $n$ is the number of tasks on the resource [15]. For a cumulative resource (resource of capacity $C > 1$), the not-first/not-last rule is more challenging because tasks may require several capacity units. Schutt et al. [14] after proving that the $O(|Sc|n^3)$-implementation (where $Sc$ is the set of distinct capacity requirements of tasks) of Nuijten [12] is incorrect and incomplete, proposed a new complete not-first/not-last algorithm which runs in $O(n^2 \log n)$ time where $n$ is the number of tasks. In [2], it is mentioned that a private communication
of Nuijten presents an $O(n^3)$ not-first/not-last algorithm. In [6], we proposed an $O(n^3)$ not-first/not-last algorithm which reaches the same fix point as the complete algorithm after at most $n$ runs. Recently, Schutt and Wolf [13] and ourselves [5] independently proposed a similar sound and incomplete not-first/not-last algorithm for cumulative resource running in $O(n^2 \log n)$ time. These algorithms do not necessary achieve the best filtering at the first run, but the filtering is improved at each subsequent run of the algorithm. The fix point is reached after at most $n$ iterations. This idea was already used for the disjunctive not-first/not-last rule [10, 15]. More recently, we improve in [7] the complete not-first/not-last algorithm of Schutt et al. [14] from $O(n^3 \log n)$ to $O(n^2 |H| \log n)$ where $|H| \leq n$ is the maximum between the number of distinct earliest completion and latest start times of tasks.

We refine in [5] our previous sound $O(n^3)$ not-first/not-last algorithm ([6], Algorithm 1) to run in $O(n^2 \log n)$. The algorithm uses the well known cumulative $\Theta$-tree data structure initially used for disjunctive filtering rules in [15] and generalized for the cumulative case in [16]. This paper is an extention of [5]. Here, first we prove that the complexity in the worst case of this algorithm for the maximal adjustment can be improved from $O(n^3 \log n)$ to $O(n^2 |H| \log n)$. The new worst case complexity of our algorithm is the same as our previous complete $O(n^2 |H| \log n)$ algorithm [7]. We then perform experimental results on benchmarks from the Project Scheduling Problem Library and the Baptiste and Le Pape data set for empirical evaluation.

The paper is organized as follows. Section 2 defines the cumulative scheduling problem, and introduces the notations used in the paper. Section 3 gives a formal definition of the not-first/not-last rules. Section 4 presents the new not-first algorithm and a proof of its soundness. In Section 5, the overall complexity of the algorithm for the maximum adjustment is proved and in Section 6 experimental results are reported. Section 7 concludes the paper and indicates our perspectives.

2. CUMULATIVE SCHEDULING PROBLEM

A resource-constrained project scheduling problem (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 start time assignment for every task satisfying the precedence and resource capacity constraints, with a makespan (i.e., the time at which all tasks are completed) at most equal to the horizon. The cumulative scheduling problem (CuSP) is a sub-problem of the RCPSP, where precedence constraints are relaxed and a single resource is considered at a time; both problems are NP-complete [1]. In a CuSP, there is a finite set of tasks or activities with fixed processing times and resource requirements. Each task has defined earliest start and latest completion times. Solving the problem consists in determining when to execute each task so that time and resource constraints are satisfied. Tasks are assumed to be processed without interruption. Formally, this problem is defined as follows:

**Definition 1** [Cumulative Scheduling Problem] — A Cumulative Scheduling Problem (CuSP) is defined by a set $T$ of tasks to be performed on a resource of capacity $C$. Each task $i \in T$ must be executed without interruption over $p_i$ units of time between an earliest start time $r_i$ (release date) and a latest end time $d_i$ (deadline). Moreover, each task requires a constant amount of resource $c_i$. It is assumed that all data are integral (or whole number). A solution of a CuSP is a schedule that assigns a start time $s_i$ to each task $i$ such that:

$$\forall i \in T: r_i \leq s_i \leq s_i + p_i \leq d_i$$ (1)

$$\forall \tau: \sum_{i \in T, s_i \leq \tau < s_i + p_i} c_i \leq C$$ (2)

The inequalities in (1) ensure that each task is assigned feasible start and end times, while (2) enforces the resource constraint.

We define the energy of a task $i$ as $e_i = c_i \cdot p_i$. This notation, along with that of earliest start and latest completion times, may be extended to non-empty sets of
tasks as follows:

\[ r_\Omega = \min_{j \in \Omega} r_j, \quad d_\Omega = \max_{j \in \Omega} d_j, \quad e_\Omega = \sum_{j \in \Omega} e_j \] (3)

By convention, if \( \Omega \) is the empty set, \( r_\Omega = +\infty \), \( d_\Omega = -\infty \), and \( e_\Omega = 0 \). Throughout the paper, we assume that for any task \( i \in T \), \( r_i + p_i \leq d_i \) and \( c_i \leq C \), otherwise the problem has no solution. We let \( n = |T| \) denote the number of tasks. \( H = \{r_i + p_i, i \in T\} \) (resp. \( H = \{d_i - p_i, i \in T\} \)) denotes the set of distinct earliest completion (resp. latest start) times of tasks and \( Sc = \{c_i, i \in T\} \) the set of distinct capacity requirements of tasks. An example of a CuSP is given in Fig. 1.

![Figure 1: A scheduling problem of 4 tasks sharing a resource of capacity C = 3.](image)

Clearly, if there exists a set of tasks \( \Omega \subseteq T \) which cannot be scheduled in the window from \( r_\Omega \) to \( d_\Omega \) without exceeding the capacity, then the CuSP has no feasible solution. Overload checking algorithms typically enforce the following relaxation of this feasibility condition, which may be computed in \( O(n \log n) \) time [16, 19]:

\[ \text{implies } \exists \text{ an overload } \geq C \]
Definition 2 (E-Feasibility) [11] — A CuSP problem is E-feasible if \( \forall \Omega \subseteq T, \Omega \neq \emptyset \)
\[
C (d_\Omega - r_\Omega) \geq e_\Omega.
\]

It is obvious that a CuSP that violates the E-feasibility condition cannot have a feasible solution. In the rest of the paper, we only consider E-feasible CuSPs.

3. The Not-First/Not-Last Rule

The not-first/not-last rule is not subsumed by the edge-finding rule and it is used to deduce that a task \( i \) cannot be the first (or the last) to be executed in \( \Omega \cup \{i\} \) all sharing the same cumulative resource. If a task \( i \) cannot be the first (resp. the last) to be executed in \( \Omega \cup \{i\} \) then the release date (resp. deadline) of task \( i \) is updated to the earliest completion time (resp. the latest start time) of the set \( \Omega \).

Definition 3 — Let \( \Omega \subseteq T \) be a set of tasks of a CuSP of capacity \( C \). The earliest completion time of a task \( i \in T \) is defined as \( ect_i := r_i + p_i \). The earliest completion time of a set of tasks \( \Omega \) is defined as \( ECT_\Omega := \min_{j \in \Omega} ect_j \) if \( \Omega \neq \emptyset \) and \( ECT_\emptyset := +\infty \).

An analogous definition is given for the latest start time of a task and of a set of tasks.

Definition 4 — Let \( \Omega \subseteq T \) be a set of tasks of a CuSP of capacity \( C \). The latest start time of a task \( i \in T \) is defined as \( lst_i := d_i - p_i \). The latest start time of a set of tasks \( \Omega \) is defined as \( LST_\Omega := \max_{j \in \Omega} lst_j \) if \( \Omega \neq \emptyset \) and \( LST_\emptyset := -\infty \).

Proposition 1 presents the not-first/not-last rule as currently known in the literature.

Proposition 1 [14] — Let \( \Omega \) be a set of tasks and let \( i \notin \Omega \).
\[
\begin{align*}
 r_i < ECT_\Omega \land e_\Omega + c_i (\min (ect_i, d_\Omega) - r_\Omega) > C (d_\Omega - r_\Omega) \Rightarrow r_i \geq ECT_\Omega \\
 LST_\Omega < d_i \land e_\Omega + c_i (d_\Omega - \max (lst_i, r_\Omega)) > C (d_\Omega - r_\Omega) \Rightarrow d_i \leq LST_\Omega
\end{align*}
\]
NOT-FIRST-NOT-LAST FILTERING ALGORITHM

The rule (NF) is called the not-first rule. It consists of updating release date of all task $i$ which cannot be the first to be executed in the set $\Omega \cup \{i\}$. A not-first algorithm is a procedure that performs all such deductions. Similarly, the rule (NL) is called the not-last rule. It consists of updating deadline of all task $i$ which cannot be the last to be executed in the set $\Omega \cup \{i\}$. We only consider not-first rule in this paper (the not-last rule is handled similarly). A complete not-first algorithm always chooses the set $\Omega$ for each task $i$ that yields the strongest update to the bound of $i$. We have the following specification of such algorithm in Definition 5.

**Definition 5** (Specification of a complete not-first algorithm) — The Not-First algorithm receives as input an E-feasible CuSP. It produces as output a vector

$$\langle LB_1, ..., LB_n \rangle$$

where

$$LB_i = \max \left( r_i, \max_{\Omega \subseteq T \setminus \{i\}} ECT_{\Omega(i)} \right)$$

and

$$\gamma(\Omega, i) \stackrel{def}{=} (r_i < ECT_{\Omega(i)} \land (e_\Omega + c_i (\min(ect_i, d_\Omega) - r_\Omega) > C(d_\Omega - r_\Omega))$$

**Example 1:** Consider the CuSP instance of Fig. 1. The not-first rule holds for task $I$ and the set $\Omega = \{A\}$ or $\Omega = \{A, B, C\}$. But the maximum adjustment occurs using the pair $(\{A\}, I)$. Indeed, $r_I = 1 < ECT_{\Omega} = ect_A = 5$ and $e_\Omega + c_I (\min(ect_I, d_\Omega) - r_\Omega) = 3 + 1 \cdot (5 - 4) = 4 > 3 = C(d_\Omega - r_\Omega)$. Hence, $LB_I = 5$.

4. A NEW NOT-FIRST ALGORITHM

In this section, we present a sound $O(n^2 \log(r_n))$ cumulative not-first algorithm. This new algorithm reaches the same fix point as the well known complete not-first/not-last algorithms [14, 7], although it may take additional iterations to do so. It is the same algorithm proposed in [5].
For a given task $i$, a complete not-first algorithm finds the strongest lower bound $LB_i$ as it is specified in Definition 5. In this section, we introduce a new not-first algorithm with $O(n^2 \log n)$ time and $O(n)$ space complexities. The new algorithm is based on the concept of the left cut of $T$ by a task introduced by Villén in [15] and the energy envelope introduced by the same author in [16].

We firstly describe how the left cut and the energy envelope can be used to check the not-first condition. Secondly, we describe how to use the cumulative $\Theta$-tree for an efficient computation of the energy envelope of a set of tasks and present the new not-first algorithm. Thirdly, soundness and the complexity of the new algorithm are proven.

**Definition 6** [5] — Let $j$ and $i$ be two tasks with $i \neq j$. The left cut of $T$ by task $j$ relatively to a task $i$ is the set of tasks $LCut(T, j, i)$ defined as follows:

\[
LCut(T, j, i) := \{k, k \in T \land k \neq i \land r_i < ect_k \land d_k \leq d_j\}. \tag{5}
\]

**Definition 7** — Let $\Theta$ be a set of tasks and $i$ be a task. The energy envelope of $\Theta$ relatively to task $i$ is the integral number $Env(\Theta, i)$ defined as follows:

\[
Env(\Theta, i) := \max_{\Omega \subseteq \Theta} \{ (C - c_i)r_\Omega + e_\Omega \}. \tag{6}
\]

A task $i \in T$ and a set of tasks $\Omega \subseteq T \setminus \{i\}$ satisfy the not-first condition if $r_i < ECT_\Omega$ and $e_\Omega + c_i(\min(ect_i, d_\Omega) - r_\Omega) > C(d_\Omega - r_\Omega)$ hold. In this paper, we prove that the not-first conditions can be checked using inequality (7) for all pair $(i, j)$ of tasks with $i \neq j$

\[
Env(LCut(T, j, i), i) > Cd_j - c_i \min(ect_i, d_j). \tag{7}
\]

**Theorem 1** — Let $i \in T$ be any task of a CuSP. There exists a set of tasks $\Omega \subseteq T \setminus \{i\}$ satisfying the condition of the not-first rule if and only if there is a task $j \in T \setminus \{i\}$ such that inequality (7) holds for tasks $i$ and $j$.

**Proof:** Let $i \in T$ be any task of a CuSP. We show both directions of the equivalence.
\( \Rightarrow \): Let us assume that there is a subset \( \Omega \subseteq T \setminus \{i\} \) such that the condition of the not-first rule holds for \( \Omega \) and \( i \), i.e. \( r_i < ECT_\Omega \) and \( e_\Omega + c_i (\text{ect}_i, d_\Omega) - r_\Omega > C(d_\Omega - r_\Omega) \). We have to show the existence of a task \( j \in T \setminus \{i\} \) that satisfies \( \text{Env}(LCut(T, j, i), i) > C d_j - c_i \text{ min}(\text{ect}_i, d_j) \).

Let \( j \in \Omega \) be a task with \( d_j = d_\Omega \). Because \( r_i < ECT_\Omega \) and \( i \notin \Omega \), it holds that \( \Omega \subseteq LCut(T, j, i) \). The inequality

\[
e_\Omega + c_i(\text{ect}_i, d_\Omega) - r_\Omega > C(d_\Omega - r_\Omega)
\]

holds since the condition \( \gamma(\Omega, i) \) holds and it is algebraically equivalent to

\[
(C - c_i)r_\Omega + e_\Omega > C d_j - c_i \text{ min}(\text{ect}_i, d_j)
\]  \hspace{1cm} (8)

According to Definition 1 and the inclusion \( \Omega \subseteq LCut(T, j, i) \) it follows that

\[
\text{Env}(LCut(T, j, i), i) \geq (C - c_i)r_\Omega + e_\Omega > C d_j - c_i \text{ min}(\text{ect}_i, d_j) \]. \hspace{1cm} (9)

Hence the inequality (7) holds for task \( j \).

\( \Leftarrow \): Let \( j \in T \setminus \{i\} \) be a task satisfying the inequality (7). We have to show the existence of a subset \( \Omega \subseteq T \setminus \{i\} \) such that the not-first conditions are satisfied.

Let \( \Omega' \subseteq LCut(T, j, i) \) be the set of tasks that determines \( \text{Env}(LCut(T, j, i), i) \), i.e. \( \text{Env}(LCut(T, j, i), i) = (C - c_i)r_\Omega' + e_\Omega' \). From \( d_{\Omega'} \leq d_j \) it follows that

\[
C d_j - c_i \text{ min}(\text{ect}_i, d_j) \geq C d_{\Omega'} - c_i \text{ min}(\text{ect}_i, d_{\Omega'}) \]. \hspace{1cm} (10)

Indeed,

- if \( \text{ect}_i \leq d_{\Omega'} \) then

\[
C d_j - c_i \text{ min}(\text{ect}_i, d_j) = C d_j - c_i \cdot \text{ect}_i
\]
\[
\geq C d_{\Omega'} - c_i \cdot \text{ect}_i
\]
\[
= C d_{\Omega'} - c_i \text{ min}(\text{ect}_i, d_{\Omega'})
\]
• if $ect_i > d_{i'}$ then

$$Cd_j - c_i \min (ect_i, d_j) \geq (C - c_i)d_j$$
$$\geq (C - c_i)d_{i'}$$
$$= Cd_{i'} - c_i \min (ect_i, d_{i'})$$

From inequality (10), it follows that

$$Env(LC\text{ut}(T, j, i), i) = (C - c_i)r_{i'} + e_{i'}$$
$$> Cd_j - c_i \min (ect_i, d_j)$$
$$\geq Cd_{i'} - c_i \min (ect_i, d_{i'})$$

and the not-first conditions hold for $\Omega'$ and $i$. ■

As in [5, 7], we assume that tasks in $T$ are sorted in non-decreasing order of deadlines. With this ordering of tasks, the set $LC\text{ut}(T, j, i)$ are quickly recomputed from the previous set i.e., if $T = \{j_1, j_2, \ldots, j_n\}$ then $LC\text{ut}(T, j_1, i) \subseteq LC\text{ut}(T, j_2, i) \subseteq \ldots \subseteq LC\text{ut}(T, j_n, i)$. Our not-first algorithm works essentially as in [5]. For each task $i \in T$, the algorithm checks the set of tasks $LC\text{ut}(T, j_{min}, i)$ of the smallest task index $j_{min}$ such that condition

$$Env(LC\text{ut}(T, j_{min}, i), i) > Cd_{j_{min}} - c_i \min (ect_i, d_{j_{min}})$$

holds. If so, it updates $r_i = ECT_{LC\text{ut}(T, j_{min}, i)}$. This adjustment is not necessarily the maximum as it is illustrated in Example 2.

**Example 2** — Let us consider the CuSP instance of Fig. 1. With $T = \{B, C, A\}$, the not-first condition is detected firstly using the set $LC\text{ut}(T, A, I) = \{B, C, A\}$. Then the release date of task $I$ is updated to $ECT_{LC\text{ut}(T, A, I)} = 3$ which is not the lowest bound as it is shown in Example 1.

To reduce the complexity of the algorithm, the sets $LC\text{ut}(T, j, i)$ are organized in a balanced binary tree called $\Theta$-tree. Tasks are represented by leaf nodes and sorted by increasing order of $r_j$ from left to right. Therefore, all classic operations (adding, removing an element in the tree) are done in $O(\log n)$ and we can integrate
the computation of the energy envelope without changing this complexity. Each node \( v \) of the tree holds the following values:

\[
e_v = e_{Leaves(v)}
\]

\[
Env_v(i) = Env(Leaves(v), i)
\]

where \( Leaves(v) \) is the set of all tasks represented by the leaves of the subtree rooted in \( v \). For a leaf node representing a task \( k \in T \), the values in the tree are set to:

\[
e_v = e_k
\]

\[
Env_v(i) = Env(\{k\}, i) = (C - c_i)r_k + e_k
\]

For internal node \( v \), these values can be computed recursively from their children nodes \( left(v) \) and \( right(v) \) as it is specified in Proposition 2.

**Proposition 2** — For a given task \( i \), for an internal node \( v \), values \( e_v \) and \( Env_v(i) \) can be computed recursively as follows:

\[
e_v = e_{left(v)} + e_{right(v)}
\]

\[
Env_v(i) = \max\{Env_{left(v)}(i) + e_{right(v)}, Env_{right(v)}(i)\}
\]
Proof: Similar to the proof of Proposition 2 in [16] by replacing $C$ in the proof of the second item by $C - c_i$. 

\[
\begin{align*}
  e_{\text{root}} &= 9 \\
  Env_{\text{root}}(I) &= 11
\end{align*}
\]

\[
\begin{align*}
  e &= 4 \\
  Env(I) &= 6
\end{align*}
\]

\[
\begin{align*}
  e &= 5 \\
  Env(I) &= 11
\end{align*}
\]

Graph 3: Calculation of $Env(\Theta, I)$ using a $\Theta$-tree for $\Theta = LCut(T, A, I) = \{A, B, C, D\}$. The values at the root are $c_{\text{root}} = 9$ and $Env_{\text{root}} = 11$. It is obvious that the not-first conditions hold since $Env_{\text{root}} = 11 > C d_A - c_i \min(ect_i, d_A) = 10$.

In Algorithm 1, the outer loop (line 3) iterates through the tasks $i \in T$ forming the possible not-first task. The inner loop (line 5) selects the task $j \in T$ that comprises the possible upper bounds for the left cut set of tasks $LCut(T, j, i)$, in non-decreasing order of deadlines. If $r_i < \text{ect}_j$ and $j \neq i$, then the task $j$ is added to the tree $\Theta = LCut(T, j, i)$ (line 7) and the earliest completion time of the set $\Theta$ is updated to reflect $ECT_\Theta$ (line 8). If the not-first condition (7) is fulfilled at line 9, then release date of task $i$ is updated to $ECT_\Theta$ (line 10). The "break" of line 11 allows to restart the detection and the update of a new task quickly. At the next iteration of the outer loop, $\Theta$ is re-initialized.

Before showing that Algorithm 1 is sound, let us prove some properties of the inner loop of line 5. Let $i \in T$ be a task, $\Theta_j := LCut(T, j, i)$ with $j \in T$ ($T$ sorted in non-decreasing order of deadlines) and $\Theta_0 := \emptyset$. 
Proposition 3 — Let \( i \in T \) be a task. In Algorithm 1, before the \( j \)th iteration of the inner loop 5, it holds that

\[
\Theta_{j-1} = \emptyset \quad \text{and} \quad \Theta_{j-1} = \emptyset \quad \text{and} \quad \Theta_{j-1} = \emptyset
\]

\[
ECT_{\Theta_{j-1}} = \min Ect \quad \text{and} \quad \min Ect = \min Ect
\]

\[
\min Ect_{\Theta_{j-1}, i} = \min Ect(\Theta, i)
\]

Proof: By induction over the array \( T = \{1, 2, \ldots, n\} \) of tasks sorted in non-decreasing order of deadlines.

Algorithm 1: Not-first algorithm in \( \mathcal{O}(n^2 \log n) \) time and \( \mathcal{O}(n) \) space

**Require:** \( T \) is an array of tasks sorted in non-decreasing order of \( d_i \)

**Private:** \( \emptyset \) : cumulative \( \emptyset \)-tree of tasks set \( LCUT(T, j, i) \) balanced by \( i \)

**Ensure:** A lower bound \( LB'_i \) is computed for the release date of each task \( i \)

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>for ( i \in T ) do</td>
</tr>
<tr>
<td>2</td>
<td>( LB'_i := r_i )</td>
</tr>
<tr>
<td>3</td>
<td>for ( i \in T ) do</td>
</tr>
<tr>
<td>4</td>
<td>( \emptyset := \emptyset ), ( \min Ect := +\infty );</td>
</tr>
<tr>
<td>5</td>
<td>for ( j \in T ) do</td>
</tr>
<tr>
<td>6</td>
<td>if ( r &lt; ect_j \land j \neq i ) then</td>
</tr>
<tr>
<td>7</td>
<td>( \emptyset := \emptyset \cup {j} );</td>
</tr>
<tr>
<td>8</td>
<td>( \min Ect := \min(\min Ect, ect_j) );</td>
</tr>
<tr>
<td>9</td>
<td>if ( Env(\emptyset, e_j) &gt; C_d_j \land \min(ect_i, d_j) ) then</td>
</tr>
<tr>
<td>10</td>
<td>( LB'_i := \max(LB'_i, \min Ect) );</td>
</tr>
<tr>
<td>11</td>
<td>break</td>
</tr>
<tr>
<td>12</td>
<td>for ( i \in T ) do</td>
</tr>
<tr>
<td>13</td>
<td>( r_i := LB'_i )</td>
</tr>
</tbody>
</table>

Basis \( j = 1 \): For a given task \( i \in T \), the inner loop is not iterated at all. From this it follows that \( \emptyset = \emptyset \) and then no earliest completion time and no energy envelope can be computed. Because \( \Theta_{j-1} = \emptyset = \emptyset \) the basis holds.
Inductive step $j \rightarrow j + 1$: The inductive hypothesis is that for all $j' < j$ before the $j'$th loop iteration the conditions (15), (16) and (17) hold. Because the tasks in the array $T$ are sorted by non-decreasing order of deadlines, it holds that $\Theta_{j-1} = \Theta_j \setminus \{j\}$. In order to show the induction step, two cases are considered: $j \in \Theta_j$ and $j \notin \Theta_j$.

First we consider the case $j \in \Theta_j$. In this case the conditions of line 6 are true for the $j$th loop iteration. According to the induction hypothesis $\Theta = \Theta_{j-1}$ is updated to $\Theta \cup \{j\}$ (line 7) and its earliest completion time is updated to reflect $\Theta_j$ (line 8). Thus, the conditions (15), (16) and (17) hold.

In the second case let $j \notin \Theta_j$. In this case the conditions of line 6 are not true. Because $j$ is not added to the tree it follows that $\Theta_j = \Theta_{j-1}$ and by the induction hypothesis the proposition holds.

Proposition 3 shows that $Env(LCut(T; j, i), i)$ and $ECT_{LCut(T; j, i)}$ are correctly computed by Algorithm 1. Combined with Theorem 1, this justifies the fact that for every task $i$, Algorithm 1 correctly detects the sets $\Omega \subseteq T \setminus \{i\}$ for which the not-first rule holds.

A complete not-first algorithm would always choose the set $\Omega$ for each task $i$ that yields the strongest update to the bound of $i$. In Theorem 2, we demonstrate that our algorithm has a slightly weaker property of soundness; that is, the algorithm updates the bounds correctly, but might not always make the strongest adjustment of the bounds at the first iteration.

**Theorem 2** — For every task $i \in T$, and given the strongest lower bound $LB_i$ as specified in Definition 5, Algorithm 1 computes some lower bound $LB'_i$, such that $r_i < LB'_i \leq LB_i$ if $r_i < LB_i$, and $LB'_i = r_i$ if $r_i = LB_i$.

**Proof:** Let $i \in T$ be any task. $LB'_i$ is initialized to $r_i$ (line 1). Because the value $LB'_i$ is only updated by $\max(LB'_i, \min Ect)$ (line 10) after each detection, it follows that $LB'_i \geq r_i$. If the equality $LB_i = r_i$ holds, then no detection is found by Algorithm 1, and thus $LB'_i = r_i$ holds from the loop at line 12. In the rest of the proof, we assume that $r_i < LB_i$.  

There exists a set of tasks $\Omega \subseteq T \setminus \{i\}$ such that the following holds:

$$LB_i = ECT_{\Omega} \land e_{\Omega} + c_i (\min(ect_i, d_{\Omega}) - r_{\Omega}) > C (d_{\Omega} - r_{\Omega}).$$

(18)

Let $j \in \Omega$ be any task with $d_j = d_{\Omega}$. According to Definition 5 of the left cut of $T$, it holds that $\Omega \subseteq \Theta_j$. According to Proposition 5 and Theorem 1, Algorithm 1 correctly detects the not-first conditions when it considers $i$ in the outer loop (line 3) and $j$ in the inner loop (line 5). From the definition of $LCut(T, j, i)$, we have $r_i < ECT_{LCut(T, j, i)} = ECT_{\Theta_j} = minEct$. Therefore, after the detection condition is fulfilled at line 9, the release date of task $i$ is updated to $LB'_i = max(LB'_i, minEct) = minEct > r_i$. Hence, Algorithm 1 is sound.

5. Overall Complexity

According to Theorem 2, Algorithm 1 will always update $r_i$ if an update is justified by the not-first rule, the even though it might be the strongest update. As there are a finite number of updating sets, Algorithm 1 reaches the same fix point as other correct not-first algorithms. Such a “lazy” approach was also used in [8, 17] to reduce the complexity of the edge-finding filtering for cumulative resources. Here, we show in Theorem 3 that in most cases Algorithm 1 finds the strongest update after at most $|I|$ propagations where $|I|$ is the maximum between the number of distinct earliest completion and latest start times of tasks.

**Theorem 3** — Let $i \in T$ be any task of an E-feasible CuSP. Let $\Omega \subseteq T \setminus \{i\}$ be a set used to perform the maximum adjustment of $r_i$ by the not-first rule. Then Algorithm 1 performs the strongest update of $r_i$ after at most $|I|$ iterations.

**Proof**: Let $i \in T$ be any task of an E-feasible CuSP. Let $\Omega \subseteq T \setminus \{i\}$ be a set used to perform the maximum adjustment of $r_i$ by the not-first rule.

Let $j \in T$ be any task with $d_j = d_{\Omega}$. Let $H_j^k := \{ect_l, l \in LCut(T, j, i)\}$ be the set of different earliest completion times of tasks of $LCut(T, j, i)$ at the $k^{th}$ iteration of Algorithm 1 (The elements in $H_j^k$ can be considered as class of tasks. An element $ect_l$ represents the set of tasks having $ect_l$ as earliest completion
time). If the maximum adjustment is not found after this iteration, then at least one element in increasing order is removed from \( H_k^j \) at the \( k + 1 \)th iteration. Indeed, if at the \( k \)th and \( k + 1 \)th iterations, \( H_k^j = H_{k+1}^j \) and the maximum adjustment is not found, then in both cases Algorithm 1 detects the not-first condition using the set of tasks \( LCcut(T, j, i) \). Therefore, at the \( k + 1 \)th iteration, no new adjustment is found, yet the maximum adjustment of the release date of task \( i \) is not reached, thus contradicting the soundness of Algorithm 1. Hence, the maximum adjustment of the release date of task \( i \) is reached after at most \( |H| \leq n \) iterations where \( |H| \) is the maximum between the number of distinct earliest completion and latest start times of tasks.

As in [6, 8], we argue that, in practice, the possibility of our algorithm using multiple propagations to find the strongest bound is not significant. In the first place, not-first/not-last rules are not idempotent; adjustment to the release dates and deadlines of the tasks are not taken into account during one iteration, so additional propagations are always required to reach a recognizable fix point. Furthermore, experimental observations reported in the next section is similar to results of [6, 8] and show that, in actual cumulative problems, there are typically a relatively small number of set of tasks that could be used to update the start time of a given task, so the number of propagations should not normally approach the worst case.

6. Experimental Results

The new not-first/not-last algorithm presented in Section 4 was implemented in C++ using the Gecode 3.4.2 [4] constraint solver. The first cumulative propagator noted N2HL for tasks of fixed duration is a sequence of four filters: the \( O(n^2) \) edge-finding algorithm from [8], the \( O(n^2|H| \log n) \) not-first/not-last algorithm from [7], overload checking and timetabling. We tested this propagator against a modified version noted N2L that substituted the new not-first/not-last filter of Section 4 for the not-first/not-last algorithm from [7].

Tests were performed on the single-mode J30 test sets of the well-established benchmark library PSPLib [9] and the library of Baptiste and Le Pape [1] (BL). The
Table 1: PSPLib and BL Tests

<table>
<thead>
<tr>
<th></th>
<th>N2HL</th>
<th></th>
<th>N2L</th>
<th></th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>solve time</td>
<td>prop</td>
<td>node</td>
<td>fail</td>
<td>solve time</td>
</tr>
<tr>
<td>(#)</td>
<td>(sec)</td>
<td>(#)</td>
<td>(#)</td>
<td>(#)</td>
<td>(sec)</td>
</tr>
<tr>
<td>J30</td>
<td>355</td>
<td>5.96</td>
<td>128129</td>
<td>8729</td>
<td>4354</td>
</tr>
<tr>
<td>BL</td>
<td>33</td>
<td>20.71</td>
<td>139133</td>
<td>13435</td>
<td>7384</td>
</tr>
</tbody>
</table>

data set J30 consists of 480 instances of 30 tasks and BL consists of 40 instances of 20 and 25 tasks respectively; these tasks require multiple shared resources, each of which was modeled with a cumulative constraint. Precedence relations were enforced as a series of linear relations between task start and end times. Branch and bound search was used to find the minimum end time for the project; variables were selected by minimum domain size, with ties broken by selecting the variable occurring in most propagators, while values were selected starting with the minimum. Tests were performed on a Intel Pentium(R) Dual Cores processor, CPU 2.8 GHz, 1 GB of RAM with a time limit of 300 seconds.

Only tests for which both propagators were able to find the best solution within 300 seconds are included in Figure 4 and Figure 5 (20 instances in which, only the propagator containing the not-first/not-last algorithm of Section 4 was able to find a solution in the time available, were discarded). In Table 1, each column “solve” reports the number of instances solved by each propagator. Column “time”, “prop”, “node” and “speedup” denotes respectively the average CPU time used to reach the optimal solution, the average number of propagations, the average number of nodes and the average speedup factor reported on instances solved by both propagators.

Our tests showed that the propagator N2L was faster in almost all test instances, with an average proportional speedup factor of 427% for J30 set and 345% for BL set. For the 388 instances solved by both filters, Figure 4 compares the runtimes of the hardest instances on J30 and BL respectively (instances with runtime greater than 1 second on the N2HL propagator).

In order to determine the difference in propagation strength between the two filters, we count the number of executions of the filtering propagators in each of the 388 solved instances. Figure 5 compares the number of propagations of the hardest instances on J30 and BL respectively. As expected, the number of propagations was different in 35 of these
Figure 4: Comparison of runtimes N2HL vs. N2L: runtimes for hard instances of j30 and BL where both methods found the best solution.

instances; however, only in 8 of those instances was the number of propagations required by the N2HL filter greater. We observe that, even in those instances where the N2L filter required a small number of propagations, the number of nodes and the number of failures (backtracks) in the search tree of the N2L algorithm were always the same as those in the N2HL tree search, implying that N2L algorithm reaches the same fix point as the N2HL filter.

7. Conclusion

In this paper, an extension of [5], we have presented a sound $O(n^2 \log n)$ not-first/not-last filtering algorithm for cumulative scheduling that reaches the same fix point as previous algorithms [7, 14], possibly after some propagations. While its complexity does not strictly dominate our previous $O(n^2 |H| \log n)$ algorithm, experimental results showed that on a standard benchmark suite our new algorithm is substantially faster. Future work will focus on finding a sound quadratic not-first/not-last algorithm, similar to the algorithm for edge finding presented in [8].
Figure 5: Comparison of runtimes N2HL vs. N2L: runtimes for hard instances of j30 and BL where both methods found the best solution.

ACKNOWLEDGEMENT

The authors would like to thank Christian Schulte for his assistance with the Gecode cumulative propagator. We would like to thank the anonymous reviewers for their comments on an early draft.

REFERENCES


3. Y. Caseau and F. Laburthe, Improved CLP scheduling with task intervals, In P.


5. R. Kameugne and L. P. Fotso, A not-first/not-last algorithm for cumulative resource in $\mathcal{O}(n^2 \log n)$, \textit{Accepted to CPDP 2010 Doctoral Program}, St Andrews, Scotland (2010).


