Time-Table Extended-Edge-Finding for the Cumulative Constraint

Pierre Ouellet and Claude-Guy Quimper

Université Laval, Québec, Canada

Abstract. We propose a new filtering algorithm for the cumulative constraint. It applies the Edge-Finding, the Extended-Edge-Finding and the Time-Tabling rules in $O(kn \log n)$ where k is the number of distinct task heights. By a proper use of tasks decomposition, it enforces the Time-Tabling rule and the Time-Table Extended-Edge-Finding rule. Thus our algorithm improves upon the best known Extended-Edge-Finding propagator by a factor of $O(\log n)$ while achieving a much stronger filtering.

1 Introduction

Scheduling problems consist of deciding when a task should start and which resource should execute it. Many side constraints can enrich the problem definition. For instance, a precedence constraint can force a task to complete before another can start. The need to cope with side constraints makes constraint programming a very attractive tool since it is handy to specify extra requirements in the problem without tweaking the scheduling algorithms provided by the constraint solver.

The CUMULATIVE constraint encodes a large variety of scheduling problems. It allows the tasks to request a portion of a cumulative resource. Tasks can execute concurrently as long as the workload is below the capacity of the resource.

There exist multiple techniques to filter the cumulative constraint. Most of these techniques are filtering rules that reason over a time interval and that deduce the relative positions between the tasks or the position relative to a given time point. Among the popular rules, there are the not-first/not-last [1], the Time-Tabling [2–5], the Edge-Finding [6, 7], the Extended-Edge-Finding [8], and the Energetic Reasoning rule [9]. The later rule dominates them all except for the not-first/not-last. Vilím [10] proposes to combine the Edge-Finding rule to the Time-Tabling rule to obtain a level of filtering greater than what is obtained by individually applying the Edge-Finding and Time-Tabling. He calls this new technique the *Timetable Edge Finding*. Schutt et al. [11] combines the technique with the use of nogoods and obtain impressive results.

We propose an algorithm that performs both Edge-Finding and Extended-Edge-Finding filtering. It is largely inspired by Vilím's Edge-Finder [6] and is mostly an extension of it. We also propose an algorithm that performs Time-Tabling and Time-Table Extended-Edge-Finding, using the pruning rules from Vilím [10]. However, our algorithm differs from [10] in three points: 1) the algorithm we present performs Time-Tabling as well as Time-Table Extended Edge-Finding; 2) when the number of distinct task heights is constant, the new algorithm runs in time $O(n \log n)$; 3) both algorithms are non-idempotent but the new algorithm guaranties to perform some filtering on all tasks for which the Edge-Finding, Extended-Edge Finding, Time-Tabling, and Time-Table Extended-Edge-Finding rules apply.

2 Preliminaries

We consider a set $\mathcal{I}$ of n non-preemptive tasks. A task $i \in \mathcal{I}$ is specified by its *earliest starting time* (est_i), its *latest completion time* (lct_i), its *processing time* (p_i), and its *height* (h_i). From the previous attributes, one can compute the *earliest completion time* (ect_i) of a task i with the relation $\mathrm{ect}_i = \mathrm{est}_i + p_i$ and its *latest starting time* (lst_i) with the relation $\mathrm{lst}_i = \mathrm{lct}_i - p_i$. The *energy* ($e_i$) of a task i is the amount of consumption of the resource during its execution and satisfies $e_i = p_i h_i$. We extend this notation to a subset of tasks $\Omega \subseteq \mathcal{I}$ as follows.

$$\mathrm{est}_\Omega = \min_{i \in \Omega} \mathrm{est}_i \qquad \mathrm{lct}_\Omega = \max_{i \in \Omega} \mathrm{lct}_i \qquad e_\Omega = \sum_{i \in \Omega} e_i \tag{1}$$

A cumulative resource is characterized by its capacity C. A task i starts at time s_i and executes during p_i units of time. The task consumes h_i units of the cumulative resource over the time period $[s_i, s_i + p_i)$. Solving a cumulative scheduling problem consists of finding, for each task $i \in \mathcal{I}$, the starting times s_i such that $\mathrm{est}_i \leq s_i \leq \mathrm{lst}_i$ and such that at any time t, the cumulative usage of the resource does not exceed C.

$$\sum_{i \in \mathcal{I} | t \in [s_i, s_i + p_i)} h_i \leq C \qquad \forall\, t \in \mathbb{Z} \tag{2}$$

Deciding whether there exists a solution to the cumulative scheduling problem is NP-Complete, even for the disjunctive case where $C = 1$.

The CUMULATIVE constraint encodes the cumulative scheduling problem (CuSP). This constraint restrains the starting times to satisfy Equation (2). It takes as parameter the vector of starting time variables, the vector of processing times, the vector of heights, and the resource capacity. The earliest starting times and latest completion times are encoded in the domains of the starting time variables by setting $\mathrm{dom}(S_i) = [\mathrm{est}_i, \mathrm{lst}_i]$.

$$\textsc{Cumulative}([S_1, \ldots, S_n], [p_1, \ldots, p_n], [h_1, \ldots, h_n], C) \tag{3}$$

2.1 Slack, E-Feasibility and Energy Envelope

For a given time interval $[a, b)$, let $\Omega = \{i \in \mathcal{I} \mid a \leq est_i \land \mathrm{lct}_i \leq b\}$, the slack ($Sl_\Omega$) is the remaining energy of the resource within the interval once all tasks in Ω are processed.

$$Sl_\Omega = C(b - a) - e_\Omega \tag{4}$$

A CuSP is said to be energy-feasible (*E-Feasible*) if it has no interval of negative slack.

The *envelope* or *energy envelope* of a task i (Env_i), is a measure of the potential consumed energy of the resource up to the completion of i. It takes into account the full resource capacity prior to the starting time of task i regardless of its effective usage. We extend the definition of the envelope to a subset of tasks $\Omega \subseteq \mathcal{I}$.

$$\mathrm{Env}_i = Cest_i + e_i \qquad \mathrm{Env}_\Omega = \max_{\Theta \subseteq \Omega}(Cest_\Theta + e_\Theta) \tag{5}$$

2.2 Edge-Finding

Edge-Finding aims at finding necessary orderings within the tasks and deducing related time-bound adjustments. The filtering usually occurs in two steps. The first step detects a relation of precedence $\Omega \lessdot i$ where $\Omega \subset \mathcal{I}$ and $i \in \mathcal{I} \setminus \Omega$. Such a precedence implies that task i finishes after all tasks in Ω are completed and is detected when the task i cannot be scheduled in the interval $[\mathrm{est}_{\Omega \cup \{i\}}, \mathrm{lct}_\Omega]$ along with the other tasks in Ω.

$$C(\mathrm{lct}_\Omega - \mathrm{est}_{\Omega \cup \{i\}}) < e_{\Omega \cup \{i\}} \Rightarrow \Omega \lessdot i \tag{6}$$

The second step consists in pruning the domain of the task i based on the detected precedence $\Omega \lessdot i$. Although several techniques exist, the goal is to deduce the availability of the resource for the task i within the interval. Nuijten [12] uses the following method. Given a set $\Theta \subseteq \Omega$, she divides and assigns the energy in e_Θ into two blocks. The first block of $(C - h_i)(\mathrm{lct}_\Theta - \mathrm{est}_\Theta)$ units of energy evenly consumes $C - h_i$ units of the resource over the time interval $[\mathrm{est}_\Theta, \mathrm{lct}_\Theta)$. The second block of energy is scheduled at its earliest time within the interval $[\mathrm{est}_\Theta, \mathrm{lct}_\Theta)$ using the remaining h_i units of resource. When this second block completes, the task i can start its execution.

$$\Omega \lessdot i \Rightarrow \mathrm{est}'_i = \max_{\Theta \subseteq \Omega} \mathrm{est}_\Theta + \left\lceil \frac{e_\Theta - (C - h_i)(\mathrm{lct}_\Theta - \mathrm{est}_\Theta)}{h_i} \right\rceil \tag{7}$$

Vilím [6] detects all precedences in $O(n \log n)$ and shows how to perform the optimal pruning in $O(kn \log n)$ where $k = |\{h_i \mid i \in \mathcal{I}\}|$ is the number of distinct task heights. By comparing tasks with *minimum slack intervals*, Kameugne et al. [7] produce a single-step quadratic algorithm. It finds all tasks that need to be adjusted according to the Edge-Finder rule and prunes them. Although their algorithm does not always deduce the best adjustment (7) on the first detection, multiple executions of their algorithm converge to the same fixed point.

2.3 Extended-Edge-Finding

The Extended-Edge-Finding rule stipulates that if the task i, when starting at its earliest time, overlaps the time interval $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega)$ and that the energy of task i over this interval plus the energy e_Ω overloads the resource, then i must finish after all tasks in Ω have completed.

$$\mathrm{est}_\Omega \in [\mathrm{est}_i, \mathrm{ect}_i) \land e_\Omega + h_i(\mathrm{ect}_i - \mathrm{est}_\Omega) > C(\mathrm{lct}_\Omega - \mathrm{est}_\Omega) \Rightarrow \Omega \lessdot i \tag{8}$$

Mercier and Van-Hentenryck [8] detect and prune the precedences in time $O(kn^2)$ where $k = |\{h_i \mid i \in \mathcal{I}\}|$ is the number of distinct task heights.

2.4 Time Tabling

Time Tabling consists of finding the necessary usage of the resource over a time interval. For a task that satisfies $\mathrm{lst}_i < \mathrm{ect}_i$, the interval $[\mathrm{lst}_i, \mathrm{ect}_i)$ determines the fixed part of the task. Let $f(\Omega, t)$ be the aggregate of the fixed parts that spans over time t by the tasks in Ω and let $f(\Omega, [a, b))$ be the aggregate of the fixed parts over the time interval $[a, b)$ by the tasks in Ω.

$$f(\Omega, t) = \sum_{i \in \Omega \mid t \in [\mathrm{lst}_i, \mathrm{ect}_i)} h_i \qquad f(\Omega, [a, b)) = \sum_{t \in [a,b)} f(\Omega, t) \tag{9}$$

If a task i cannot complete before time t and $h_i + f(\mathcal{I} \setminus \{i\}, t) > C$, then the task i must start after time t.

$$\mathrm{ect}_i > t \land C < h_i + f(\mathcal{I} \setminus \{i\}, t) \Rightarrow \mathrm{est}'_i > t \tag{10}$$

Figure 1 depicts the Time-Tabling rule. Letort et al. [5] introduce a *sweep* technique that iterates over time and gradually enlarges the aggregate while pruning the tasks. Their method is later improved [13] and copes with very large sets of tasks. Beldiceanu et al. [4] propose an original technique reasoning over slack using a relation with the problem of rectangles placement.

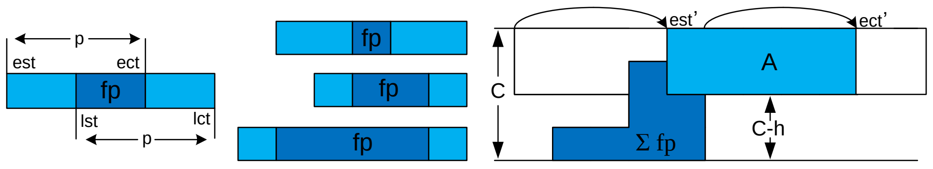

Fig. 1. A task with a fixed part, all tasks with a fixed part, the aggregate of the fixed parts and the Time-Tabling rule applied to task A.

2.5 Time-Table Extended-Edge-Finding

Recent efforts [10, 11] enhanced the Edge-Finding and Extended-Edge-Finding rules by taking into account the necessary usage of the resource due to fixed parts. The Time-Table Extended Edge-Finding combines the techniques of the Time-Tabling, the Edge-Finding, and the Extended-Edge-Finding. Let e^f_Ω be the energy of the tasks in Ω plus the fixed energy of the tasks in $\mathcal{I} \setminus \Omega$ spent within the interval $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega)$.

$$e^f_\Omega = e_\Omega + f(\mathcal{I} \setminus \Omega, [\mathrm{est}_\Omega, \mathrm{lct}_\Omega)) \tag{11}$$

Substituting e_Ω by e^f_Ω in (6) and (8) leads to the Time-Table Extended-Edge-Finding rules. Substituting e_Θ by e^f_Θ in (7) gives the new adjustment rule.

2.6 The Cumulative Tree and the Overload Checking Test

The algorithm we propose utilizes a cumulative tree similar to those introduced by Vilím [6, 14, 15]. The cumulative tree is an essentially complete binary tree with n leaves. Its main purpose is to compute the time interval $[a, b)$ that optimizes functions of the form $f(a, b, \Omega, \Lambda)$. Its leaves from left to right are associated to the tasks sorted in non-decreasing order of earliest starting time (est). The leaf of task i is labeled $\{i\}$ and their association holds throughout the execution of the algorithm. When the algorithm moves a task from a set to another, values in its associated leaf are re-initialized accordingly and the functions are updated from the leaf up to the root in $O(\log n)$. This data structure has proven very effective in particular for the Overload Checking that tests the E-feasibility. We illustrate in the following.

The function to optimize is the envelope of the subset $\Omega \subseteq \mathcal{I}$. From Equation (5).

$$\mathrm{Env}_\Omega = \max_{\Theta \in \Omega}(Cest_\Theta + e_\Theta)$$

The algorithm initializes all tasks as member of Ω. It iterates over every task j in decreasing order of lct_j. The algorithm ends an iteration by moving task j from Ω to Λ triggering a sequence of updates from its associated leaf. It results in the root holding the maximum envelope value of all intervals $[est_\Theta, \mathrm{lct}_\Theta) \subseteq [est_\Omega, \mathrm{lct}_\Omega)$ where $\mathrm{lct}_\Theta = \mathrm{lct}_\Omega = \mathrm{lct}_j$ at the beginning of any iteration. If $\mathrm{Env}_\Omega > C\ \mathrm{lct}_j$, the algorithm detects an overload.

To achieve the computation (see Figure 2), an envelope value and an energy value are required in every nodes. For a leaf $\{i\}$, these values are those of its corresponding task $e_{\{i\}} = p_i h_i$ and $\mathrm{Env}_{\{i\}} = Cest_i + e_i$ when $i \in \Omega$. They are set to zero when the task is moved to Λ. For the inner nodes v, the values are computed from the ones held by their left (l) and right (r) children as follows.

$$e_v = e_l + e_r \qquad\qquad \mathrm{Env}_v = \max\{\mathrm{Env}_l + e_r\,,\ \mathrm{Env}_r\}$$

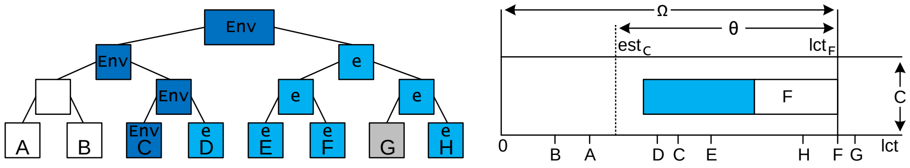

Fig. 2. A cumulative tree with its leaves sorted in increasing order of est (left) and a schematic representation of the cumulative resource with the time axis labeled with the lct (right). The algorithm moved task G to Λ at the previous iteration. It now iterates over task F. At this point, all tested intervals are upper bounded by lct_F. In this instance, the maximum envelope value is induced by the leaf associated to task C. The right part of the figure shows the optimal interval $[est_\Theta, \mathrm{lct}_\Theta) = [est_C, \mathrm{lct}_F)$. It is composed by the set of tasks $\{C, D, E, F, H\}$. The left part of the figure shows all the values that are cumulated up to the root resulting in the optimal value.

3 New Filtering Rules

The algorithm we present enforces the Edge-Finding and Extended-Edge-Finding rules to filter the lower bound of the starting time variables. A symmetric algorithm can filter the upper bounds. It proceeds by detecting any surplus of energy within a time interval $[est_\Omega, \mathrm{lct}_\Omega)$ should a task i start at its *earliest starting time* est_i. If the surplus is positive, the algorithm detects that task i cannot start at time est_i and performs the exact adjustment to the lower bound of task i that erases the surplus.

We consider two cases where the Edge-Finding rule applies. The *weak case* occurs when the Edge-Finding rule (6) applies and $\mathrm{ect}_i < \mathrm{lct}_\Omega$ holds. We denote this rule EF^w. The *strong case* occurs when $\mathrm{ect}_i \geq \mathrm{lct}_\Omega$ and leads to the strong Edge-Finding rule denoted EF^s. The weak and strong cases also apply to the Extended-Edge-Finding rule (8) and leads to the two detection rules EEF^w and EEF^s[1]. In all cases, the Edge-Finding rules apply when $\mathrm{est}_\Omega < \mathrm{est}_i$ and the Extended-Edge-Finding rules apply when $\mathrm{est}_\Omega \geq \mathrm{est}_i$.

When one of these four rules detects a *surplus*, we denote by $\sigma_{EF^w}(i, \Omega)$, $\sigma_{EEF^w}(i, \Omega)$, $\sigma_{EF^s}(i, \Omega)$, and $\sigma_{EEF^s}(i, \Omega)$ the extra energy requirement in the time interval $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega)$ should task i start at time est_i.

$$\sigma_{EF^w}(i, \Omega) = e_{\Omega \cup \{i\}} - C(\mathrm{lct}_\Omega - \mathrm{est}_\Omega) \tag{12}$$
$$\sigma_{EEF^w}(i, \Omega) = e_\Omega + h_i(\mathrm{ect}_i - \mathrm{est}_\Omega) - C(\mathrm{lct}_\Omega - \mathrm{est}_\Omega) \tag{13}$$
$$\sigma_{EF^s}(i, \Omega) = e_\Omega + h_i(\mathrm{lct}_\Omega - \mathrm{est}_i) - C(\mathrm{lct}_\Omega - \mathrm{est}_\Omega) \tag{14}$$
$$\sigma_{EEF^s}(i, \Omega) = e_\Omega - (C - h_i)(\mathrm{lct}_\Omega - \mathrm{est}_\Omega) \tag{15}$$

These quantities are used to combine the detection and the adjustment rules into a single rule that adjusts the earliest starting time of task i. In the weak case ($\mathrm{ect}_i < \mathrm{lct}_\Omega$), we obtain these two rules.

$$EF^w : \mathrm{est}_i \geq \mathrm{est}_\Omega \wedge \sigma_{EF^w}(i, \Omega) > 0 \Rightarrow \mathrm{est}'_i = \mathrm{lct}_\Omega - p_i + \left\lceil \frac{\sigma_{EF^w}(i, \Omega)}{h_i} \right\rceil$$

$$EEF^w : \mathrm{est}_i < \mathrm{est}_\Omega \wedge \sigma_{EEF^w}(i, \Omega) > 0 \Rightarrow$$
$$\mathrm{est}'_i = \mathrm{lct}_\Omega - (\mathrm{ect}_i - \mathrm{est}_\Omega) + \left\lceil \frac{\sigma_{EEF^w}(i, \Omega)}{h_i} \right\rceil$$

In the strong case ($\mathrm{ect}_i \geq \mathrm{lct}_\Omega$), we have this adjustment rule for the Edge-Finding

$$EF^s : \mathrm{est}_i \geq \mathrm{est}_\Omega \wedge \sigma_{EF^s}(i, \Omega) > 0 \Rightarrow \mathrm{est}'_i = \mathrm{est}_i + \left\lceil \frac{\sigma_{EF^s}(i, \Omega)}{h_i} \right\rceil$$

and the following one for the Extended-Edge-Finding

$$EEF^s : \mathrm{est}_i < \mathrm{est}_\Omega \wedge \sigma_{EEF^s}(i, \Omega) > 0 \Rightarrow \mathrm{est}'_i = \mathrm{est}_\Omega + \left\lceil \frac{\sigma_{EEF^s}(i, \Omega)}{h_i} \right\rceil$$

[1] The rules EF^w, EEF^w, EF^s, and EEF^s respectively represents the cases *inside*, *left*, *right*, and *through* in [10].

We show that these new adjustment rules are identical to the adjustment rule (7) when the relation $\Theta = \Omega$ holds. The case when $\Theta \subset \Omega$ is handled later.

Lemma 1. *The rules EF^w, EEF^w, EF^s, and EEF^s are equivalent to the adjustment rule* (7) *when $\Theta = \Omega$.*

Proof. The adjustment for the rule EF^s is

$$\begin{aligned}\mathrm{est}'_i &= \mathrm{est}_i + \left\lceil \frac{e_\Omega + h_i(\mathrm{lct}_\Omega - \mathrm{est}_i) - C(\mathrm{lct}_\Omega - \mathrm{est}_\Omega)}{h_i} \right\rceil \\ &= \mathrm{est}_\Omega + \left\lceil \frac{e_\Omega - (C - h_i)(\mathrm{lct}_\Omega - \mathrm{est}_\Omega)}{h_i} \right\rceil\end{aligned}$$

which is equivalent to rule (7) when $\Theta = \Omega$. The adjustment for the rule EEF^s is

$$\mathrm{est}'_i = \mathrm{est}_\Omega + \left\lceil \frac{e_\Omega - (C - h_i)(\mathrm{lct}_\Omega - \mathrm{est}_\Omega)}{h_i} \right\rceil$$

which is equivalent to rule (7) when $\Theta = \Omega$. The adjustment for the rule EF^w is

$$\begin{aligned}\mathrm{est}'_i &= \mathrm{lct}_\Omega - p_i + \left\lceil \frac{e_\Omega + e_i - C(\mathrm{lct}_\Omega - \mathrm{est}_\Omega)}{h_i} \right\rceil = \mathrm{lct}_\Omega + \left\lceil \frac{e_\Omega - C(\mathrm{lct}_\Omega - \mathrm{est}_\Omega)}{h_i} \right\rceil \\ &= \mathrm{est}_\Omega + \left\lceil \frac{e_\Omega - (C - h_i)(\mathrm{lct}_\Omega - \mathrm{est}_\Omega)}{h_i} \right\rceil\end{aligned}$$

which is equivalent to rule (7) when $\Theta = \Omega$. The adjustment for the rule EEF^w is

$$\begin{aligned}\mathrm{est}'_i &= \mathrm{lct}_\Omega - (\mathrm{ect}_i - \mathrm{est}_\Omega) + \left\lceil \frac{e_\Omega + h_i(\mathrm{ect}_i - \mathrm{est}_\Omega) - C(\mathrm{lct}_\Omega - \mathrm{est}_\Omega)}{h_i} \right\rceil \\ &= \mathrm{lct}_\Omega + \left\lceil \frac{e_\Omega - C(\mathrm{lct}_\Omega - \mathrm{est}_\Omega)}{h_i} \right\rceil\end{aligned}$$

This form was already proved equivalent to rule (7) when $\Theta = \Omega$. □

We show that successively applying, in no particular order, the rules EF^w, EEF^w, EF^s, and EEF^s leads to the same fixed point as the adjustment rule (7).

Lemma 2. *After applying the rules EF^w and EEF^w, the inequality $\mathrm{ect}'_i \geq \mathrm{lct}_\Omega$ holds, where ect'_i is the new earliest completion time of task i.*

Proof. After applying the rule EF^w, we obtain $\mathrm{ect}'_i = \mathrm{lct}_\Omega + \left\lceil \frac{\sigma_{EF^w}(i,\Omega)}{h_i} \right\rceil$. Since $\sigma_{EF^w}(i,\Omega) > 0$, we have $\mathrm{ect}'_i > \mathrm{lct}_\Omega$. The same applies for the rule EEF^w. □

Lemma 2 ensures that when there are tasks for which the weak rules EF^w and EEF^w apply, after the adjustment of the rules, only the strong rules EF^s and EEF^s can apply.

Lemma 3. *Successively applying the adjustment rules EF^w, EEF^w, EF^s, and EEF^s leads to the same fixed point obtained by using the adjustment rule* (7).

Proof. Let Θ be the set that maximizes the expression in (7). Since Lemma 1 covers the case where $\Theta = \Omega$, we suppose that $\Theta \subset \Omega$. In the strong case, we have the inequalities $\mathrm{lct}_\Theta \leq \mathrm{lct}_\Omega \leq \mathrm{ect}_i$. In the weak case, Lemma 2 ensures that these inequalities also hold after applying the rules EF^w or EEF^w. Therefore, we only need to check whether the rules EF^s and EEF^s can be applied with the set of tasks Θ. Since the set Θ leads to an adjustment, the numerator in (7) is positive which implies $e_\Theta > (C - h_i)(\mathrm{lct}_\Theta - \mathrm{est}_\Theta)$. If $\mathrm{est}_i < \mathrm{est}_\Theta$ then the rule EEF^s applies and leads to the same filtering as rule (7).

Suppose that $\mathrm{est}_i \geq \mathrm{est}_\Theta$ and that the adjustment rule (7) prunes the earliest starting time est_i further. Then this inequality holds.

$$\mathrm{est}_i < \mathrm{est}_\Theta + \frac{e_\Theta - (C - h_i)(\mathrm{lct}_\Theta - \mathrm{est}_\Theta)}{h_i} \tag{16}$$

This is equivalent to $0 < e_\Theta + h_i(\mathrm{lct}_\Theta - \mathrm{est}_i) - C(\mathrm{lct}_\Theta - \mathrm{est}_\Theta)$. Therefore, $\sigma_{EF^s}(i, \Theta) > 0$ and the rule EF^s prunes the est at the same position as rule (7) does. Consequently, after adjusting est_i, either $\Theta = \Omega$ and the adjustment is equivalent to the rule (7) or $\Theta \subset \Omega$ and the rules EF^s and EEF^s can still be applied in a future iteration. □

4 A New Extended-Edge-Finding Algorithm

We present a new algorithm that performs the Extended-Edge-Finding. Algorithm 1 is largely based on Vilím's algorithm [6] for the Edge-Finding of the cumulative constraint and its cumulative tree data structure. We broaden the scope of the cumulative tree with two more sets, Ψ and Γ and substitute Ω for Θ to go along our notation. Therefore, the algorithm uses a cumulative $\Omega, \Lambda, \Psi, \Gamma$ tree. These four sets are different status of the tasks during the execution of the algorithm and serve computational purposes. The mechanic of the cumulative tree is illustrated in Section 2.6.

An essentially complete binary tree of $|\mathcal{I}|$ leaves is built, with leaves from left to right associated to the tasks sorted in non-decreasing order of est, breaking ties on the smallest lct. The algorithm iterates on heights in $\{h_i \mid i \in \mathcal{I} \wedge \mathrm{ect}_i < \mathrm{lct}_i\}$ in arbitrary order, with h being the current height. These operations occur within an iteration.

The cumulative tree is initialized with all its tasks in Ω. It iterates through the tasks in non-increasing order of latest completion time (lct). We say that j is the current task. Thus, lct_j is the upper bound of all optimized intervals at the current iteration.

The algorithm partitions the tasks $\mathcal{I}$ into four sets: Γ is the set of excluded tasks, $\Omega = \{i \in \mathcal{I} \setminus \Gamma \mid \mathrm{lct}_i \leq \mathrm{lct}_j\}$ is the set of unprocessed tasks, $\Lambda = \{i \in \mathcal{I} \setminus (\Omega \cup \Gamma) \mid h_i = h, \mathrm{ect}_i < \mathrm{lct}_j\}$ is the set of processed tasks of height h with earliest completion time smaller than lct_j, and $\Psi = \{i \in \mathcal{I} \setminus (\Omega \cup \Gamma) \mid h_i = h, \mathrm{ect}_i \geq \mathrm{lct}_j\}$ is the set of processed tasks of height h with earliest completion time greater than or equal to lct_j. As it iterates through the tasks, the current latest completion time lct_j changes and might result in moving tasks from Λ to Ψ. At any time, a task can move from Λ and Ψ to the set of excluded tasks. Those are tasks that are ignored for the rest of the iteration. At the end of the iteration, the task j is removed from Ω and added to Λ if $h_j = h \wedge \mathrm{ect}_j < \mathrm{lct}_j$, otherwise, the task cannot be further filtered and is added to Γ.

The algorithm utilizes the cumulative tree to optimize the surplus functions (12) to (15) and performs an overload check. Whenever a detection applies, the corresponding task is pruned according to the adjustment rule and then moved to Γ. Then, the algorithm updates the nodes from the leaf associated to the pruned task up to the root and checks for an other detection. To efficiently compute the functions, eleven values are held in the nodes. Some of these values are function of the horizon $\mathrm{Hor} = \max_{i \in \mathcal{I}} \mathrm{lct}_i$, i.e. the latest time when a task can complete. For a leaf node v, these values are.

$$e_v = \begin{cases} e_i & \text{if } i \in \Omega \\ 0 & otherwise \end{cases} \qquad \mathrm{Env}_v = \begin{cases} C\,\mathrm{est}_i + e_i & \text{if } i \in \Omega \\ -\infty & \text{otherwise} \end{cases} \tag{17}$$

$$\mathrm{Env}^h_v = \begin{cases} (C-h)\,\mathrm{est}_i + e_i & \text{if } i \in \Omega \\ -\infty & \text{otherwise} \end{cases} \qquad e^{\Lambda}_v = \begin{cases} e_i & \text{if } i \in \Lambda \\ -\infty & \text{otherwise} \end{cases} \tag{18}$$

$$\mathrm{Env}^{\Lambda} = \begin{cases} C\,\mathrm{est}_i + e_i & \text{if } i \in \Lambda \\ -\infty & \text{otherwise} \end{cases} \qquad \mathrm{ex}^{\Lambda}_v = \begin{cases} h\,\mathrm{ect}_i & \text{if } i \in \Lambda \\ -\infty & \text{otherwise} \end{cases} \tag{19}$$

$$e^{\Psi}_v = \begin{cases} h(\mathrm{Hor} - \mathrm{est}_i) & \text{if } i \in \Psi \\ -\infty & \text{otherwise} \end{cases} \qquad \mathrm{Env}^{\Psi}_v = \begin{cases} C\,\mathrm{est}_i + e^{\Psi} & \text{if } i \in \Psi \\ -\infty & \text{otherwise} \end{cases} \tag{20}$$

$$\mathrm{ex}^{\Psi}_v = \begin{cases} h_i\,\mathrm{Hor} & \text{if } i \in \Psi \\ -\infty & \text{otherwise} \end{cases} \tag{21}$$

$$\mathrm{Envx}^{\Lambda}_v = -\infty \qquad \mathrm{Envx}^{\Psi}_v = -\infty \tag{22}$$

For an inner node v, its left child and right child are denoted $\mathrm{left}(v)$ and $\mathrm{right}(v)$. These values are computed recursively as follows.

$$e_v = e_{\mathrm{left}(v)} + e_{\mathrm{right}(v)} \tag{23}$$

$$\mathrm{Env}_v = \max(\mathrm{Env}_{\mathrm{left}(v)} + e_{\mathrm{right}(v)}, \mathrm{Env}_{\mathrm{right}(v)}) \tag{24}$$

$$\mathrm{Env}^h_v = \max(\mathrm{Env}^h_{\mathrm{left}(v)} + e_{\mathrm{right}(v)}, \mathrm{Env}^h_{\mathrm{right}(v)}) \tag{25}$$

$$e^{\Lambda}_v = \max(e^{\Lambda}_{\mathrm{left}(v)} + e_{\mathrm{right}(v)}, e_{\mathrm{left}(v)} + e^{\Lambda}_{\mathrm{right}(v)}) \tag{26}$$

$$\mathrm{Env}^{\Lambda}_v = \max(\mathrm{Env}^{\Lambda}_{\mathrm{left}(v)} + e_{\mathrm{right}(v)}, \mathrm{Env}_{\mathrm{left}(v)} + e^{\Lambda}_{\mathrm{right}(v)}, \mathrm{Env}^{\Lambda}_{\mathrm{right}(v)}) \tag{27}$$

$$\mathrm{ex}^{\Lambda}_v = \max(\mathrm{ex}^{\Lambda}_{\mathrm{left}(v)}, \mathrm{ex}^{\Lambda}_{\mathrm{right}(v)}) \tag{28}$$

$$e^{\Psi}_v = \max(e^{\Psi}_{\mathrm{left}(v)} + e_{\mathrm{right}(v)}, e_{\mathrm{left}(v)} + e^{\Psi}_{\mathrm{right}(v)}) \tag{29}$$

$$\mathrm{Env}^{\Psi}_v = \max(\mathrm{Env}^{\Psi}_{\mathrm{left}(v)} + e_{\mathrm{right}(v)}, \mathrm{Env}_{\mathrm{left}(v)} + e^{\Psi}_{\mathrm{right}(v)}, \mathrm{Env}^{\Psi}_{\mathrm{right}(v)}) \tag{30}$$

$$\mathrm{ex}^{\Psi}_v = \max(\mathrm{ex}^{\Psi}_{\mathrm{left}(v)}, \mathrm{ex}^{\Psi}_{\mathrm{right}(v)}) \tag{31}$$

$$\mathrm{Envx}^{\Lambda}_v = \max(\mathrm{Envx}^{\Lambda}_{\mathrm{left}(v)} + e_{\mathrm{right}(v)}, \mathrm{ex}^{\Lambda}_{\mathrm{left}(v)} + \mathrm{Env}^h_{\mathrm{right}(v)}, \mathrm{Envx}^{\Lambda}_{\mathrm{right}(v)}) \tag{32}$$

$$\mathrm{Envx}^{\Psi}_v = \max(\mathrm{Envx}^{\Psi}_{\mathrm{left}(v)} + e_{\mathrm{right}(v)}, \mathrm{ex}^{\Psi}_{\mathrm{left}(v)} + \mathrm{Env}^h_{\mathrm{right}(v)}, \mathrm{Envx}^{\Psi}_{\mathrm{right}(v)}) \tag{33}$$

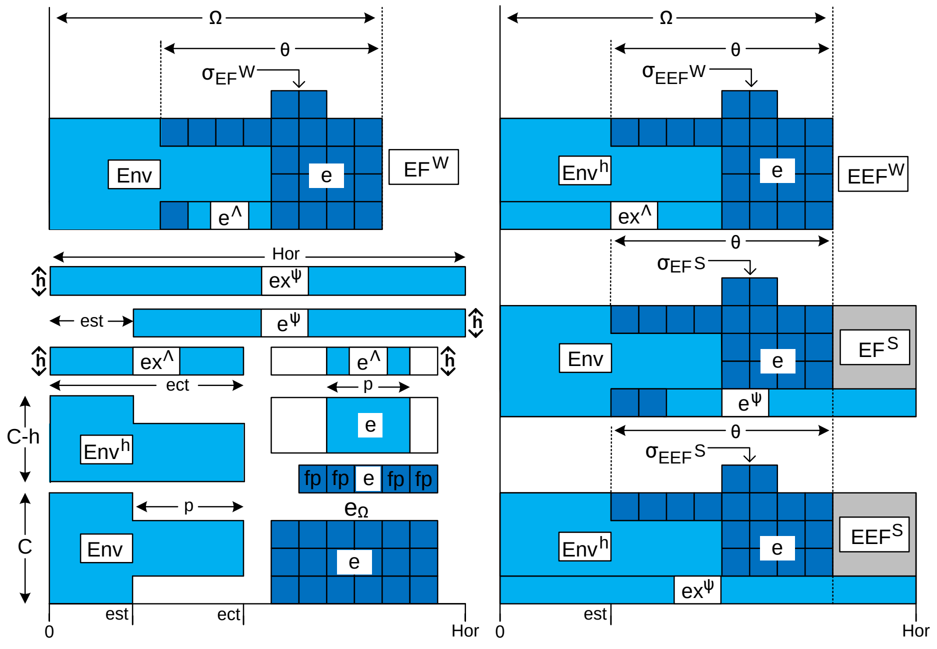

Fig. 3. Geometric illustration of the values cumulated by the tree, the four filtering rules and their detected surplus. The blue squares depict the cumulated energy of all tasks in Ω. The figure shows the optimal interval $[\mathrm{est}_\Theta, \mathrm{lct}_\Theta)$ within $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega)$. All four rules are a combination of the energy of a task $i \notin \Omega$ and an optimal envelope. In this figure, each rule detects a surplus of 2 units of energy.

At the root of the tree, four values are particularly important and have the following equivalences. We use these relations to rewrite the conditions of the Edge-Finding and Extended-Edge-Finding rules.

$$\mathrm{Env}^\Lambda = \max_{\substack{\Theta \subseteq \Omega \\ \mathrm{lct}_\Theta = \mathrm{lct}_\Omega}} \max_{\substack{i \in \Lambda \\ \mathrm{est}_\Theta \le \mathrm{est}_i}} C\,\mathrm{est}_\Theta + e_\Theta + e_i \tag{34}$$

$$\mathrm{Env}^\Psi = \max_{\substack{\Theta \subseteq \Omega \\ \mathrm{lct}_\Theta = \mathrm{lct}_\Omega}} \max_{\substack{i \in \Psi \\ \mathrm{est}_\Theta \le \mathrm{est}_i}} C\,\mathrm{est}_\Theta + e_\Theta + h(\mathrm{Hor} - \mathrm{est}_i) \tag{35}$$

$$\mathrm{Envx}^\Lambda = \max_{\substack{\Theta \subseteq \Omega \\ \mathrm{lct}_\Theta = \mathrm{lct}_\Omega}} \max_{\substack{i \in \Lambda \\ \mathrm{est}_i \le \mathrm{est}_\Theta}} (C - h)\,\mathrm{est}_\Theta + e_\Theta + h\,\mathrm{ect}_i \tag{36}$$

$$\mathrm{Envx}^\Psi = \max_{\substack{\Theta \subseteq \Omega \\ \mathrm{lct}_\Theta = \mathrm{lct}_\Omega}} \max_{\substack{i \in \Psi \\ \mathrm{est}_i \le \mathrm{est}_\Theta}} (C - h)\,\mathrm{est}_\Theta + e_\Theta + h\,\mathrm{Hor} \tag{37}$$

The functions $\sigma_{EF^w}(i, \Omega)$, $\sigma_{EEF^w}(i, \Omega)$, $\sigma_{EF^s}(i, \Omega)$, and $\sigma_{EEF^s}(i, \Omega)$ can be optimized using the functions above.

$$\max_{\substack{\Theta \subseteq \Omega \\ \mathrm{lct}_\Theta = \mathrm{lct}_\Omega}} \max_{\substack{i \in \Lambda \\ \mathrm{est}_\Theta \le \mathrm{est}_i}} \sigma_{EF^w}(i, \Theta) = \mathrm{Env}^{\Lambda} - C\,\mathrm{lct}_j \tag{38}$$

$$\max_{\substack{\Theta \subseteq \Omega \\ \mathrm{lct}_\Theta = \mathrm{lct}_\Omega}} \max_{\substack{i \in \Psi \\ \mathrm{est}_\Theta \le \mathrm{est}_i}} \sigma_{EF^s}(i, \Theta) = \mathrm{Env}^{\Psi} - C\,\mathrm{lct}_j - h(\mathrm{Hor} - \mathrm{lct}_j) \tag{39}$$

$$\max_{\substack{\Theta \subseteq \Omega \\ \mathrm{lct}_\Theta = \mathrm{lct}_\Omega}} \max_{\substack{i \in \Lambda \\ \mathrm{est}_i \le \mathrm{est}_\Theta}} \sigma_{EEF^w}(i, \Theta) = \mathrm{Envx}^{\Lambda} - C\,\mathrm{lct}_j \tag{40}$$

$$\max_{\substack{\Theta \subseteq \Omega \\ \mathrm{lct}_\Theta = \mathrm{lct}_\Omega}} \max_{\substack{i \in \Psi \\ \mathrm{est}_i \le \mathrm{est}_\Theta}} \sigma_{EEF^s}(i, \Theta) = \mathrm{Envx}^{\Psi} - C\,\mathrm{lct}_j - h(\mathrm{Hor} - \mathrm{lct}_j) \tag{41}$$

Using the above relations, Algorithm 1 computes the surplus and applies the rules EF^w, EF^s, EEF^w, and EEF^s accordingly. The for loop on line 1 iterates $k = |\{h_i \mid i \in \mathcal{I}\}|$ times. Each time the repeat loop on line 2 executes, a task moves out from the set Λ or Ψ which can happen only once for each task. Such an operation triggers the update of the cumulative tree in time $\Theta(\log n)$. The total running time complexity is therefore $O(kn \log n)$.

5 Task Decomposition and Time-Tabling

We show how to decompose a problem with n tasks into a problem with at most $5n$ tasks. This decomposition facilitates the design of a new algorithm for the Time-Tabling. It also allows to perform the Time-Table Extended-Edge-Finding not by changing the Algorithm 1, but rather by changing its input. Task decomposition is a technique also used by Schutt et al. [1] and Vilím [10].

The tasks in $\mathcal{I}$ are decomposed into two sets of tasks: the depleted tasks $\mathcal{T}$ and the fixed tasks $\mathcal{F}$. For every task i such that $\mathrm{lst}_i < \mathrm{ect}_i$, there is a fixed energy of height h_i in the interval $[\mathrm{lst}_i, \mathrm{ect}_i)$. We replace the task $i \in \mathcal{I}$ by the task $i' \in \mathcal{T}$ with $\mathrm{est}_{i'} = \mathrm{est}_i$, $\mathrm{lct}_{i'} = \mathrm{lct}_i$, $p_{i'} = p_i - \mathrm{ect}_i + \mathrm{lst}_i$, and $h_{i'} = h_i$. If $\mathrm{lst}_i \ge \mathrm{ect}_i$, we create a task $i' \in \mathcal{T}$ that is a copy of the original task i. Let Z be the set of all time points est_i, lst_i, ect_i, and lct_i. We consider two consecutive time points a and b in Z with positive fixed energy, i.e. $f(\mathcal{I}, [a, b)) > 0$. We create a *fixed task* $f \in \mathcal{F}$ with $\mathrm{est}_f = a$, $\mathrm{lct}_f = b$, $p_f = b - a$, $h_f = f(\mathcal{I}, a)$. This task has no choice but to execute at its earliest starting time.

Since $|Z| \le 4n$, there are fewer than $4n$ fixed tasks and the decomposition has fewer than $5n$ tasks. Two distinct tasks $f_1, f_2 \in \mathcal{F}$ produce two disjoint intervals $[\mathrm{est}_{f_1}, \mathrm{lct}_{f_1})$ and $[\mathrm{est}_{f_2}, \mathrm{lct}_{f_2})$. Figure 4 depicts this transformation.

5.1 Task Decomposition Algorithm

Algorithm 2 takes as input the set of original tasks $\mathcal{I}$ and returns the set of depleted tasks $\mathcal{T}$ and the set of fixed tasks $\mathcal{F}$. The algorithm has a running time complexity of $O(n \log n)$. Indeed, the dimension of vector r is at most $4n$ and requires $O(n \log n)$ to sort. The function `IndexOf` can be implemented with a binary search with time complexity $O(\log n)$ and is called at most n times. The first and second for loop have a time complexity of $O(n \log n)$ and $O(n)$ for a total of $O(n \log n)$.

Algorithm 1: ExtendedEdgeFinder($\mathcal{I}$)

```
   Hor $\leftarrow \max_{i\in\mathcal{I}} \mathrm{lct}_i$;
1  **for** $h \in \{h_i \mid i \in \mathcal{I} \wedge \mathrm{ect}_i < \mathrm{lct}_i\}$ **do**
       $\Omega \leftarrow \mathcal{I}$;
       $\Lambda \leftarrow \emptyset$;
       $\Psi \leftarrow \emptyset$;
       **for** $j \in \mathcal{I}$ *in non-increasing order of* $\mathrm{lct}_j$ **do**
           **if** $\mathrm{Env} > C\,\mathrm{lct}_j$ **then** Fail;
           $\Delta \leftarrow \{i \in \Lambda \mid \mathrm{ect}_i \geq \mathrm{lct}_j\}$;
           $\Lambda \leftarrow \Lambda \setminus \Delta$;
           $\Psi \leftarrow \Psi \cup \Delta \setminus \{i \in \Psi \mid \mathrm{est}_i \geq \mathrm{lct}_j\}$;
2          **repeat**
               $\sigma(EF^w) \leftarrow \mathrm{Env}^{\Lambda} - C\,\mathrm{lct}_j$;
               $\sigma(EEF^w) \leftarrow \mathrm{Envx}^{\Lambda} - C\,\mathrm{lct}_j$;
               $\sigma(EF^s) \leftarrow \mathrm{Env}^{\Psi} - C\,\mathrm{lct}_j - h(\mathrm{Hor} - \mathrm{lct}_j)$;
               $\sigma(EEF^s) \leftarrow \mathrm{Envx}^{\Psi} - C\,\mathrm{lct}_j - h(\mathrm{Hor} - \mathrm{lct}_j)$;
               $m \leftarrow \max\{\sigma(EEF^w), \sigma(EEF^s), \sigma(EF^w), \sigma(EF^s), \}$;
               **if** $\sigma(EEF^w) = m > 0$ **then**
                   Let $i \in \Lambda$ be the unique task whose value $\mathrm{ex}^{\Lambda}$ is used for the
                   computation of $\mathrm{Envx}^{\Lambda}$;
                   Let $k \in \Omega$ be the unique task whose value $\mathrm{est}_k$ is used for the
                   computation of $\mathrm{Env}^{h}$;
                   $\mathrm{est}'_i \leftarrow \mathrm{lct}_j - (\mathrm{ect}_i - \mathrm{est}_k) + \left\lceil \frac{\sigma(EEF^w)}{h_i} \right\rceil$;
                   $\Lambda \leftarrow \Lambda \setminus \{i\}$;
               **else if** $\sigma(EEF^s) = m > 0$ **then**
                   Let $i \in \Psi$ be the task with smallest est whose value $\mathrm{ex}^{\Psi}$ is used for the
                   computation of $\mathrm{Envx}^{\Psi}$;
                   Let $k \in \Omega$ be the unique task whose value $\mathrm{est}_k$ is used for the
                   computation of $\mathrm{Env}^{h}$;
                   $\mathrm{est}'_i \leftarrow \mathrm{est}_k + \left\lceil \frac{\sigma(EEF^s)}{h_i} \right\rceil$;
                   $\Psi \leftarrow \Psi \setminus \{i\}$;
               **else if** $\sigma(EF^w) = m > 0$ **then**
                   Let $i \in \Lambda$ be the unique task whose value $e_v^{\Lambda}$ is used for the computation
                   of $\mathrm{Env}^{\Lambda}$;
                   $\mathrm{est}'_i \leftarrow \mathrm{lct}_j - p_i + \left\lceil \frac{\sigma(EF^w)}{h_i} \right\rceil$;
                   $\Lambda \leftarrow \Lambda \setminus \{i\}$;
               **else if** $\sigma(EF^s) = m > 0$ **then**
                   Let $i \in \Psi$ be the unique task whose value $e^{\Psi}$ is used for the computation
                   of $\mathrm{Env}^{\Psi}$;
                   $\mathrm{est}'_i \leftarrow \mathrm{est}_i + \left\lceil \frac{\sigma(EF^s)}{h_i} \right\rceil$;
                   $\Psi \leftarrow \Psi \setminus \{i\}$;
           **until** $m \leq 0$ ;
           **if** $h_j = h \wedge \mathrm{ect}_j < \mathrm{lct}_j$ **then** $\Lambda \leftarrow \Lambda \cup \{j\}$;
           $\Omega \leftarrow \Omega \setminus \{j\}$;
```

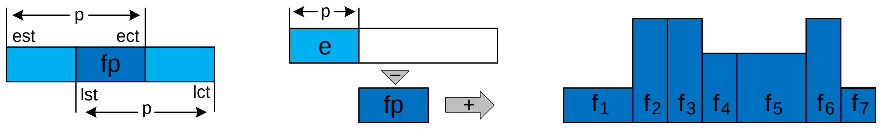

Fig. 4. A task with a fixed part, the same task after depletion of its fixed energy, and an energy aggregate turned into a set of fixed tasks $\mathcal{F}$.

Algorithm 2: TimeTableTaskDecomposition($\mathcal{I}$)

```
Create the sorted vector $r = \{\text{est}_i, \text{ect}_i, \text{lst}_i, \text{lct}_i\}$ for all $i \in \mathcal{I}$ without duplicates;
Create the null vector $c$ of dimension $|r|$;
$\mathcal{T} \leftarrow \emptyset, \mathcal{F} \leftarrow \emptyset$;
for $i \in \mathcal{I}$ do
    if $\text{ect}_i > \text{lst}_i$ then
        $a \leftarrow \texttt{IndexOf}(\text{lst}_i, r)$;
        $b \leftarrow \texttt{IndexOf}(\text{ect}_i, r)$;
        $c[a] \leftarrow c[a] + h_i$;
        $c[b] \leftarrow c[b] - h_i$;
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{\texttt{Task}(\text{est} = \text{est}_i, \text{lct} = \text{lct}_i, h = h_i, p = p_i - \text{ect}_i + \text{lst}_i)\}$;
    else
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{\texttt{Task}(\text{est} = \text{est}_i, \text{lct} = \text{lct}_i, h = h_i, p = p_i\}$;
for $l = 1..|r| - 1$ do
    $c[l] \leftarrow c[l] + c[l-1]$;
    if $c[l-1] > C$ then Failure;
    if $c[l-1] > 0$ then
        $\mathcal{F} \leftarrow \mathcal{F} \cup \{\texttt{Task}(\text{est} = r[l-1], \text{lct} = r[l], h = c[l-1], p = r[l] - r[l-1])\}$;
return $(\mathcal{T}, \mathcal{F})$
```

5.2 Time-Tabling Algorithm

Algorithm 3 sorts the tasks $\mathcal{T}$ in non-decreasing heights and the fixed tasks $\mathcal{F}$ in non-increasing heights. It maintains, using an AVL tree, a set S of time intervals in which the unprocessed tasks in $\mathcal{T}$ cannot execute concurrently with the fixed tasks. The set S grows as the algorithm iterates through $\mathcal{T}$. While processing the task $i' \in \mathcal{T}$, if there exists an interval $[a, b) \subseteq S$ such that $\text{est}_{i'} < b$ and $\text{est}_{i'} + p_{i'} > a$ then the algorithm retrieves the original task $i \in \mathcal{I}$ associated to i' and performs the pruning $\text{est}_i \leftarrow \min(\text{lst}_i, b)$. When $\text{lst}_i < b$, the earliest starting time is set to lst_i to force the task to start at the beginning of its fixed part. The AVL tree finds the interval $[a, b)$ in $O(\log |\mathcal{F}|)$. Sorting the tasks require $O(|\mathcal{T}| \log |\mathcal{T}|)$ and $O(|\mathcal{F}| \log |\mathcal{F}|)$. Since $|\mathcal{T}|, |\mathcal{F}| \in O(n)$, the overall complexity is $O(n \log n)$.

5.3 Time-Table Extended-Edge-Finding

We use the decomposition to perform Time-Table Extended-Edge-Finding. After reaching a fixed point with Algorithm 2 and 3, we pass the tasks $\mathcal{T} \cup \mathcal{F}$ as input to Algo-

Algorithm 3: FilterTimeTabling($\mathcal{T}, \mathcal{F}$)

```
Sort the fixed tasks $\mathcal{F}$ in non-increasing order of heights;
$S \leftarrow \{\infty\}$; $j \leftarrow 0$;
for $i' \in \mathcal{T}$ in non-decreasing order of height do
    while $j < |\mathcal{F}| \wedge h_{\mathcal{F}[j]} > C - h_{i'}$ do
        $S \leftarrow S \cup [\mathrm{est}_{\mathcal{F}[j]}, \mathrm{lct}_{\mathcal{F}[j]})$;
        $j \leftarrow j + 1$;
    $b \leftarrow \min\{b \notin S \mid b - 1 \in S \wedge \mathrm{est}_{i'} < b\}$;
    $a \leftarrow \min\{a \in S \mid [a, b) \subseteq S\}$;
    if $\mathrm{est}_{i'} + p_{i'} > a$ then
        if $\mathrm{lst}_i \geq \mathrm{ect}_i$ then $\mathrm{est}_i \leftarrow b$;
        else $\mathrm{est}_i \leftarrow \min(\mathrm{lst}_i, b)$;
```

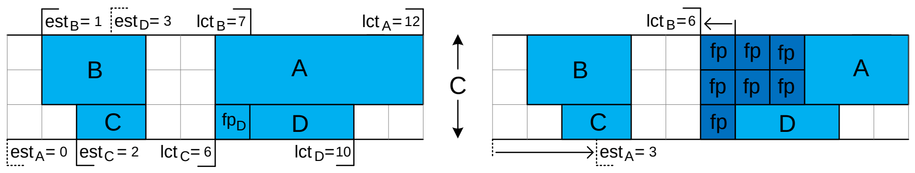

Fig. 5. The left part depicts a CuSP with 4 tasks. The upper and lower parts of the time axis indicates the earliest starting times and the latest completion times. The grid determines the energy units. The processing times and heights are to scale. By not taking into account the fixed part of task D, neither the Time-Tabling rule nor the Extended-Edge-Finding rule can deduct a pruning. A decomposition of task D leads to two consecutive updates. The rule EEF^w updates the lower bound of Task A to 3 which creates 6 new units of fixed energy. Then, the Time-Tabling rule adjusts the upper bound of task B to 6. The right part depicts the resulting CuSP.

rithm 1. Since the fixed tasks will not be filtered, the for loop on line 1 can restrict the iterations over the heights of the tasks in $\mathcal{T}$. When the earliest starting time of task $i' \in \mathcal{T}$ is filtered to time t, we filter the est of the original task $i \in \mathcal{I}$ to time $\mathrm{est}_i \leftarrow \min(t, \mathrm{lst}_i)$. This ensures to perform Time-Table Edge-Finding in time $O(kn \log n)$. Figure 5 shows an example where a task is filtered by Time-Table Extended-Edge-Finding.

6 Experiments

We tested the different versions of the algorithm with the PSLIB benchmark (Projection Scheduling Problem Library) [16]. More precisely, we solved instances of the single-mode resource-constrained project scheduling problem (SMRCPSP). Those instances are based on series of tasks that can be completed before a given horizon limit. A number of resources is given with varying capacities of production. Each task has a duration and an amount of a specific resources used during its execution. Each task also has a list of other tasks, its successors, that can be started only after this task is completed.

The model is based on two constraints. We use a precedence constraint to ensure the order of the successors is respected and we use a cumulative constraint for each resource that ensures the execution of the tasks does not overload the resources. We set the makespan to the best known value reported for the benchmark. We use a binary variable to enforce a precedence between each relevant pair of tasks. We branch on the precedence constraints that involve the tasks with the most similar resource consumptions and the largest processing times.

We used the CP solver Choco version 2.1.5 on a computer with a AMD Athlon(tm) II P340 Dual-Core running at 2.20GHz. We ran simultaneously 2 experiments, one per core. We used the cumulative constraint available in Choco that performs Time-Tabling [5] and Extended-Edge-Finding [8] that we denote *Choco*. We denote the Algorithm 1 combined with the Algorithm 3 *EEF+TT* and the Time-Table Extended-Edge-Finding *TTEEF*. Table 1 reports the results.

Benchmark			Choco			EEF+TT			TTEEF		
n	#instances	time out	solved	bt	time	solved	bt	time	solved	bt	time
30	480	10	364	8757	223	377	8757	50	377	8379	54
60	480	20	332	3074	1527	340	3074	269	341	2861	291
90	480	50	321	5024	5522	327	5024	857	329	4635	913

Table 1. Experimental results. Section *Benchmark* reports the number of tasks n, the number of instances, and the time out (in seconds) used for the experiment. For each filtering algorithm, we report the number of instances solved (*solved*). We report the cumulative number of backtracks (*bt*) and the cumulative time (*time*) required to solve all instances that are commonly solved by the three algorithms.

Choco and EEF+TT produce the same number of backtracks since they offer the same filtering. However, EEF+TT is significantly faster than Choco and solves more instances. TTEEF is slightly slower in time than EEF+TT but solves few more instances in fewer backtracks.

7 Conclusion

We presented three new algorithms that filter the CUMULATIVE constraint. The first algorithm is an Extended-Edge-Finder with a time complexity of $O(kn \log n)$. The second filtering algorithm performs Time-Tabling in time $O(n \log n)$. The third algorithm performs Time-Table Extended-Edge-Finding in time $O(kn \log n)$. These new algorithms proved to be very efficient in practice offering a fast and strong filtering.

References

1. Schutt, A., Wolf, A.: A new $O(n^2 \log n)$ 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). (2010) 445–459
2. Aggoun, A., Beldiceanu, N.: Extending chip in order to solve complex scheduling and placement problems. Mathematical and Computer Modelling **17**(7) (1993)
3. Baptiste, P., Pape, C.L.: Constraint propagation techniques for disjunctive scheduling: The preemptive case. In: Proceedings of the 12th European Conference on Artificial Intelligence (ECAI 1996). (1996)
4. Beldiceanu, N., carlsson, M., Poder, E.: New filtering for the cumulative constraint in the context of non-overlapping rectangles. In: Proceedings of the 5th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CPAIOR 2008). (2008) 21–35
5. Beldiceanu, N., Carlsson, M.: A new multi-resource cumulatives constraint with negative heights. In: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP 2002). (2002) 63–79
6. Vilím, P.: Edge finding filtering algorithm for discrete cumulative resources in $O(kn \log n)$. In: Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming (CP 2009). (2009) 802–816
7. Kameugne, R., Fotso, L., Scott, J., Ngo-Kateu, Y.: A quadratic edge-finding filtering algorithm for cumulative resource constraints. In: Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP 2011). (2011) 478–492
8. Mercier, L., Van Hentenryck, P.: Edge finding for cumulative scheduling. INFORMS Journal on Computing **20**(1) (2008) 143–153
9. Baptiste, P., Pape, C.L., Nuijten, W.: Constraint-Based Scheduling. Kluwer Academic Publishers (2001)
10. Vilím, P.: Timetable edge finding filtering algorithm for discrete cumulative resources. In: Proceedings of the 8th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2011). (2011) 230–245
11. Schutt, A., Feydy, T., Stuckey, P.J.: Explaining time-table-edge-finding propagation for the cumulative resource constraint. In: Proceedings of the 10th International Conference on Integration of AI and OR Techniques and Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2013). (2013)
12. Nuijten, W.: Time and Resource Constrained Scheduling. PhD thesis, Eindhoven University of Technology (1994)
13. Letort, A., Beldiceanu, N., Carlsson, M.: A scalable sweep algorithm for the cumulative constraint. In: Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP 2012). (2012) 439–454
14. Vilím, P.: $O(n \log n)$ filtering algorithms for unary reource constraint. In: Proceedings of the 1st International conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CPAIOR 2004). (2004) 335–347
15. Vilím, P.: Max energy filtering algorithm for discrete cumulative resources. In: Proceedings of the 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2009). (2009) 294–308
16. Kolisch, R., Sprecher, A.: Psplib - a project scheduling library. European Journal of Operational Research **96** (1996) 205–216 http://webserver.wi.tum.de/psplib/.