Edge Finding Filtering Algorithm for Discrete Cumulative Resources in $\mathcal{O}(kn \log n)$

Petr Vilím

ILOG S.A. an IBM Company, 9, rue de Verdun, BP 85
F-94253 Gentilly Cedex, France
petr_vilim@cz.ibm.com

Abstract. This paper presents new Edge Finding algorithm for discrete cumulative resources, i.e. resources which can process several activities simultaneously up to some maximal capacity C. The algorithm has better time complexity than the current version of this algorithm: $\mathcal{O}(kn \log n)$ versus $\mathcal{O}(kn^2)$ where n is number of activities on the resource and k is number of distinct capacity demands. Moreover the new algorithm is slightly stronger and it is able to handle optional activities. The algorithm is based on the Θ-tree – a binary tree data structure which already proved to be very useful in filtering algorithms for unary resource constraints.
Note: This version of the paper differs from the original: it fixes several mistakes (mostly typos). All fixes are marked by footnotes and they include author of the fix.
Version: October 28$^{\text{th}}$ 2010

1 Introduction

Nowadays, constraint based scheduling engines like IBM ILOG CP-Optimizer [1] allow to describe and solve very complex scheduling problems involving a variety of different constraints. This paper describes a filtering algorithm called Edge Finding for one of them – for discrete cumulative resource constraint.

Let us demonstrate the problem on a simple example on Figure 1. Note that this example may be just a small part of much more complex problem.

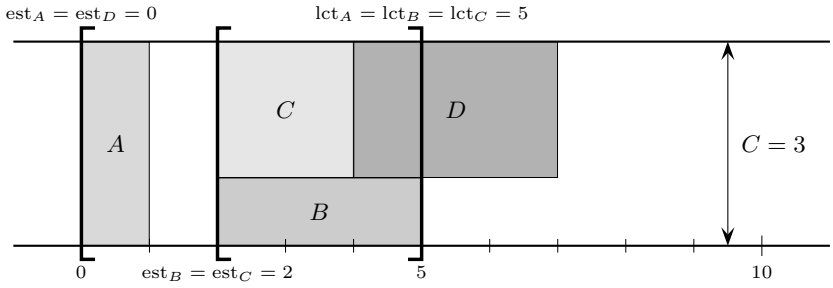

Fig. 1. An example: est$_D$ can be updated from 0 to 4.

There are three equivalent workers (a resource with capacity $C = 3$) who must perform four different activities $T = \{A, B, C, D\}$. Activity A requires all three workers ($\mathrm{c}_A = 3$) for one hour ($\mathrm{p}_A = 1$), activity B requires only one worker ($\mathrm{c}_B = 1$) but for 3 hours ($\mathrm{p}_B = 3$) and remaining two activities C and D require two workers each ($\mathrm{c}_C = \mathrm{c}_D = 2$), activity C for 2 hours ($\mathrm{p}_C = 2$) and activity D for 3 hours ($\mathrm{p}_D = 3$). Moreover the earliest possible starting time of activities A and D is zero ($\mathrm{est}_A = \mathrm{est}_D = 0$), for activities B and C it is 2 ($\mathrm{est}_B = \mathrm{est}_C = 2$). Latest possible completion time (deadline) for activities A, B and C is 5 ($\mathrm{lct}_A = \mathrm{lct}_B = \mathrm{lct}_C = 5$), activity D does not have a deadline ($\mathrm{lct}_D = \infty$).

Looking closely to the problem we can see that there is no way for D to start before 4. Therefore we can update est_D from 0 to 4 and this way limit the search space of the problem. The rest of this paper describes the algorithm (called Edge Finding) which performs such an update.

Note that there are also filtering algorithms for discrete cumulative resource other than Edge Finding. For example Timetable propagation [2], Not-First/Not-Last [3], Energetic Reasoning [2], Max Energy propagation [4] or Extended Edge Finding [5]. However Timetable and Edge Finding are the ones which are used most of the time.

Let us quickly review existing Edge Finding algorithms for discrete cumulative resources. To the author's knowledge the current state-of-the-art algorithm can be found in [5]. In this paper Luc Mercier and Pascal Van Hentenryck proved that the original cumulative Edge Finding algorithm with time complexity $\mathcal{O}(n^2)$ in [2] is incomplete, and therefore they designed new (complete) algorithm with time complexity $\mathcal{O}(kn^2)$.

This paper further improves the algorithm [5] in several aspects:

- Θ-tree data structure improves time complexity from $\mathcal{O}(kn^2)$ to $\mathcal{O}(kn \log n)$.
- Better usage of relation "ends before end" makes the filtering a little bit stronger. There are cases when the new algorithm propagates while the old one does not (Section 6.2).
- The algorithm can be easily adapted to handle optional activities. Although propagation of optional activities can be further improved [6], it is already pretty strong. We will show how to handle optional activities at the end of the paper (Section 9).
- The algorithm is based on modified Edge Finding rules which are more suitable for propagation. We provide a proof that the new rules are not weaker than the original ones (Section 8).

Like algorithm [5] the new algorithm has two phases:

Detection phase tries to discover necessary relative positions of activities on the resource. The result of this phase is a partial knowledge of a relation "ends before end" (see later), which will be used in the next phase. For the example on Figure 1 the algorithm in this phase detects that activity D must end after the end of $\{A, B, C\}$.

Adjustment phase utilizes results of the previous phase to improve temporal bounds of activities – earliest start times and latest completion times.

In comparison with algorithm [5] the time complexities of both phases are improved. For detection phase from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$, for adjustment phase from $\mathcal{O}(kn^2)$ to $\mathcal{O}(kn \log n)$. For simplicity we will describe each phase separately.

We present only the algorithm to update earliest start times (not latest completion times) because the algorithm for update of latest completion times is symmetrical.

Note also that Edge Finding algorithm is not idempotent and therefore it is usually repeated until a fixpoint is reached.

2 Basic Notation

Let us formalize the notation already used in the introduction. The input of the algorithm is a discrete cumulative resource with capacity $C \in \mathbb{N}^+$ and a set of activities T ($|T| = n$) which must be processed by the resource. Each activity $i \in T$ is characterized by the following attributes:

- earliest possible start time $\mathrm{est}_i \in \mathbb{N}$,
- latest possible completion time $\mathrm{lct}_i \in \mathbb{N}$,
- processing time (duration) $\mathrm{p}_i \in \mathbb{N}$ – a constant,
- required capacity $\mathrm{c}_i \in \mathbb{N}$ – a constant $\mathrm{c}_i \leq C$.

Activities are not preemptive, that is, if an activity i starts at time t it must continue without interruption until time $t + \mathrm{p}_i$ where it ends. During the whole processing from t to $t + \mathrm{p}_i$ it requires capacity c_i from the resource. At any time, the total capacity required from the resource cannot exceed the maximum capacity C. We define k to be number of distinct capacity demands $k = |\{c_l,\ l \in T\}|$.

Values est_i and lct_i can change – they can be updated by other filtering algorithms or by the Edge Finding algorithm itself. Therefore the input of the Edge Finding algorithm are current bounds est_i and lct_i, the output are new (updated) bounds.

Note that at any time the following inequality must hold for every activity i:

$$\mathrm{est}_i + \mathrm{p}_i \leq \mathrm{lct}_i$$

If it does not hold then the problem does not have any solution and the propagation ends (a *fail*).

From the processing time and required capacity we can compute an energy of an activity i:

$$\mathrm{e}_i = \mathrm{c}_i \, \mathrm{p}_i$$

The energy corresponds to the area occupied by the activity on the resource when depicted like on Figure 1.

The notation for earliest start time, latest completion time and energy can be easily extended for a set of activities $\Omega \subseteq T$:

$$\begin{aligned} \mathrm{est}_\Omega &= \min\{\mathrm{est}_i,\ i \in \Omega\} \\ \mathrm{lct}_\Omega &= \max\{\mathrm{lct}_i,\ i \in \Omega\} \\ \mathrm{e}_\Omega &= \sum_{i \in \Omega} \mathrm{e}_i \end{aligned}$$

3 Earliest Completion Time, Energy Envelope

A critical role in the algorithm plays a computation of possible earliest completion time of a set of intervals $\Theta \subseteq T$. This computation was already described in detail in [4], therefore here we only quickly repeat the main idea. We defined a lower bound $\mathrm{Ect}(\Theta)$ of earliest completion time of a set of activities $\Theta \subseteq T$ as:

$$\mathrm{Ect}(\Theta) = \left\lceil \frac{\mathrm{Env}(\Theta)}{C} \right\rceil$$

where $\mathrm{Env}(\Theta)$ is so-called *energy envelope* of set Θ:

$$\mathrm{Env}(\Theta) = \max_{\Omega \subseteq \Theta} \{C\,\mathrm{est}_\Omega + \mathrm{e}_\Omega\} \tag{1}$$

3.1 Cumulative Θ-tree

The paper [4] also describes how to compute $\mathrm{Env}(\Theta)$. The idea is to organize set Θ in a balanced binary tree which we call cumulative Θ-tree. Activities are represented by leaf nodes and sorted by est_i from left to right. Each node v of the tree holds the following values:

$$\mathrm{e}_v = \mathrm{e}_{\mathrm{Leaves}(v)} \tag{2}$$

$$\mathrm{Env}_v = \mathrm{Env}\left(\mathrm{Leaves}\left(v\right)\right) \tag{3}$$

Where $\mathrm{Leaves}(v)$ is a set of all activities represented by leaves of the subtree rooted in v.

Figure 2 shows a Θ-tree for $\Theta = \{A, B, C, D\}$ from Figure 1. Notice that the energy envelope of the represented set Θ is equivalent to the value Env of the root node. We can conclude that $\mathrm{Ect}\left(\{A, B, C, D\}\right) = \lceil 16/3 \rceil = 6$.

For a leaf node v representing an activity $i \in T$ the values in the tree are set to:

$$\mathrm{e}_v = \mathrm{e}_i \qquad\qquad \mathrm{Env}_v = \mathrm{Env}\left(\{i\}\right)$$

For internal nodes v these values can be computed recursively from their children nodes $\mathrm{left}(v)$ and $\mathrm{right}(v)$ as shown in the following proposition.

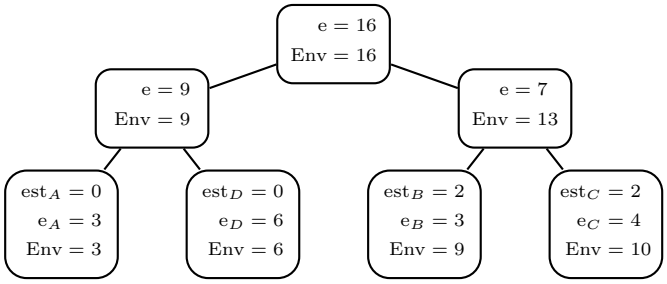

Fig. 2. An example of a Θ-tree for $\Theta = \{A, B, C, D\}$ from Figure 1.

Proposition 1. *For an internal node v, values e_v and Env_v can be computed by the following recursive formulas:*

$$\mathrm{e}_v = \mathrm{e}_{\mathrm{left}(v)} + \mathrm{e}_{\mathrm{right}(v)} \tag{4}$$

$$\mathrm{Env}_v = \max\left\{\mathrm{Env}_{\mathrm{left}(v)} + \mathrm{e}_{\mathrm{right}(v)},\ \mathrm{Env}_{\mathrm{right}(v)}\right\} \tag{5}$$

Proof. See [4]. □

Thanks to formulas (4) and (5), computation of values e_v and Env_v can be integrated within standard operations with balanced binary trees without changing their usual time complexities.

4 Relation "Ends before End"

Before going into details about Edge Finding, let us introduce a notion of *ends before end*. We say that an activity j ends before the end of an activity i (denoted by $j \lessdot i$) if in all solutions $\mathrm{end}(j) \leq \mathrm{end}(i)$. The goal of the Edge Finding algorithm is to discover as much of the relation $\lessdot$ as possible and use it to update temporal bounds. Until a solution is found the relation "ends before end" is known only partially. Therefore in the following we will use the notation $j \lessdot i$ in the sense "it is known that in all solutions j ends before the end of i".

The notation for "ends before end" can be extended also to sets of activities:

$$\forall \Omega \subset T,\ \forall i \in T \setminus \Omega: \quad \Omega \lessdot i \ \Leftrightarrow (\forall j \in \Omega:\ j \lessdot i)$$

5 Edge Finding: Detection Rule

Let us start by definition of a *left cut of T by activity j*:

$$\mathrm{LCut}(T, j) = \{l,\ l \in T\ \&\ \mathrm{lct}_l \leq \mathrm{lct}_j\}$$

To detect the relation $\lessdot$ we will use the following rule:

$$\forall j \in T,\ i \in T \setminus \mathrm{LCut}\,(T, j):$$
$$(\mathrm{Ect}\,(\mathrm{LCut}\,(T, j) \cup \{i\}) > \mathrm{lct}_j\ \Rightarrow\ \mathrm{LCut}(T, j) \lessdot i)$$

The idea of this rule follows. The set $\mathrm{LCut}(T, j)$ must be processed before lct_j. So if there is not enough time to process i together with $\mathrm{LCut}(T, j)$ then the activity i must end after $\mathrm{LCut}(T, j)$. Note that this rule is different from the original Edge Finding rule. We will show that this new rule is not weaker later in Section 8.

The rule above can be rewritten using energy envelope into a form more suitable for the algorithm:

$$\forall j \in T,\, i \in T \setminus \mathrm{LCut}\,(T, j) : \\ (\mathrm{Env}\,(\mathrm{LCut}\,(T, j) \cup \{i\}) > C\,\mathrm{lct}_j \;\Rightarrow\; \mathrm{LCut}(T, j) \lessdot i) \quad \text{(EF1)}$$

Our goal is to discover as much of the relation $\lessdot$ as possible. Therefore for each activity $i \in T$ we are looking for an activity $j \in T$ with maximal[1] lct_j such that $\mathrm{LCut}(T, j) \lessdot i$ can be detected by the rule (EF1). This is the task for the first part of the algorithm.

6 Detection Algorithm

Notice that once we prove by (EF1) that $\mathrm{LCut}\,(T, j) \lessdot i$ then it is pointless to evaluate the rule (EF1) for the same activity i and any $j' \in T$ such that $\mathrm{lct}'_j \leq \mathrm{lct}_j$ because it cannot bring any new information ($\mathrm{LCut}(T, j') \subseteq \mathrm{LCut}(T, j)$).

The algorithm is similar to the Edge Finding algorithm for unary resource [7]. We iterate over activities j in non-increasing order by lct_j and we maintain a set $\Lambda \subseteq T \setminus \mathrm{LCut}\,(T, j)$ of all activities i for which we still did not find a set which must end before end of i. In each step of the algorithm we check whether there is some activity $i \in \Lambda$ such that the rule (EF1) proves that $\mathrm{LCut}\,(T, j) \lessdot i$. In other words, we test whether the following inequality holds:

$$\max_{i \in \Lambda} \{\mathrm{Env}\,(\mathrm{LCut}\,(T, j) \cup \{i\})\} > C\,\mathrm{lct}_j$$

- If it holds then we find the responsible activity $i \in \Lambda$ and conclude that $\mathrm{LCut}\,(T, j) \lessdot i$. Therefore we can remove i from Λ.
- If it does not hold then we move activity j into Λ and continue by next activity j (because there is no activity i such that $\mathrm{LCut}(j) \lessdot i$ can be proved by (EF1)).

To formalize the algorithm let us define:

$$\mathrm{Env}\,(\Theta, \Lambda) = \max_{i \in \Lambda} \{\mathrm{Env}\,(\Theta \cup \{i\})\}$$

Although we did not show yet how to compute $\mathrm{Env}\,(\Theta, \Lambda)$ we can already present the resulting Algorithm 1.1. The result of the computation is the array `prec` which has the following meaning:

$$\forall i \in T : \quad \{l,\, l \in T \;\&\; \mathrm{lct}_l \leq \mathtt{prec}\,[i]\} \lessdot i$$

[1] Maximality of lct_j assures that for any other $j' \in T$ satisfying $\mathrm{LCut}(T, j') \lessdot i$ by (EF1) we have $\mathrm{LCut}(T, j') \subseteq \mathrm{LCut}(T, j)$.

Algorithm 1.1. Edge Finding: Detection

```
1  for i ∈ T do
2    prec[i] := −∞;
3  Θ := T;
4  Λ := ∅;
5  for j ∈ T in non-increasing order of lct_j do begin
6    while Env(Θ, Λ) > C lct_j do begin
7      i := activity from Λ responsible for Env(Θ, Λ);
8      prec[i] := lct_j; // means: LCut(T, j) ≪ i
9      Λ := Λ \ {i};
10   end;
11   Θ := Θ \ {j};
12   Λ := Λ ∪ {j};
13 end;
```

In the algorithm, $\Theta = \mathrm{LCut}(T, j)$ unless there are duplicities in lct_j (the algorithm is correct even with such duplicities). In the following we will concentrate on the computation of $\mathrm{Env}\,(\Theta, \Lambda)$ and the proof that the algorithm 1.1 has time complexity $\mathcal{O}(n \log n)$.

6.1 Computation of Env (Θ, Λ)

The idea is to extend cumulative Θ-tree into Θ-Λ-tree in a similar way it was done for unary resource in [7]. The cumulative Θ-Λ-tree is a balanced binary tree where each leaf represents one activity from the set Θ or Λ. Leaves are sorted from left to right according to est_i. Each node of the tree holds the following values:

$$\begin{aligned}
\mathrm{e}_v &= \mathrm{e}_{\mathrm{Leaves}(v)\cap\Theta} \\
\mathrm{e}^{\Lambda}_v &= \mathrm{e}_{\mathrm{Leaves}(v)\cap\Theta} + \max_{i\in\mathrm{Leaves}(v)\cap\Lambda}\{\mathrm{e}_i\} \\
\mathrm{Env}_v &= \mathrm{Env}\,(\mathrm{Leaves}\,(v)\cap\Theta) \\
\mathrm{Env}^{\Lambda}_v &= \mathrm{Env}\,(\mathrm{Leaves}\,(v)\cap\Theta, \mathrm{Leaves}\,(v)\cap\Lambda)
\end{aligned}$$

Notice that $\mathrm{Env}\,(\Theta, \Lambda)$ is equivalent to Env^{Λ}_v in the root node. For an example of the cumulative Θ-Λ-tree see Figure 3.

For a leaf node v an activity $i \in \Theta \cup \Lambda$ these values are set to:

$$\mathrm{e}_v = \begin{cases} \mathrm{e}_i & \text{if } i \in \Theta \\ 0 & \text{if } i \in \Lambda \end{cases} \qquad \mathrm{e}^{\Lambda}_v = \begin{cases} -\infty & \text{if } i \in \Theta \\ \mathrm{e}_i & \text{if } i \in \Lambda \end{cases}$$

$$\mathrm{Env}_v = \begin{cases} C\,\mathrm{est}_i + \mathrm{e}_i & \text{if } i \in \Theta \\ -\infty & \text{if } i \in \Lambda \end{cases} \qquad \mathrm{Env}^{\Lambda}_v = \begin{cases} -\infty & \text{if } i \in \Theta \\ C\,\mathrm{est}_i + \mathrm{e}_i & \text{if } i \in \Lambda \end{cases}$$

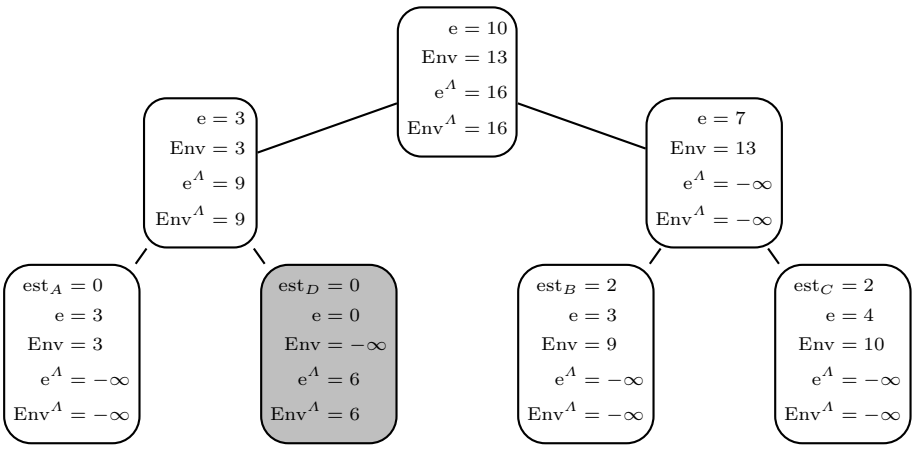

Fig. 3. An example of a Θ-Λ-tree for $\Theta = \mathrm{LCut}(T, A) = \{A, B, C\}$ and $\Lambda = \{D\}$ from Figure 1. We see that $\mathrm{Env}(\Theta, \Lambda) = 16$ which is more than $C\,\mathrm{lct}_A = 15$ and therefore $\{A, B, C\} \lessdot D$.

For internal nodes v these values are computed recursively from their children nodes $\mathrm{left}(v)$ and $\mathrm{right}(v)$:

Proposition 2. *For an internal node v values e_v, e^{Λ}_v, Env_v and Env^{Λ}_v can be computed by the following formulas:*

$$\mathrm{e}_v = \mathrm{e}_{\mathrm{left}(v)} + \mathrm{e}_{\mathrm{right}(v)} \tag{6}$$

$$\mathrm{e}^{\Lambda}_v = \max\left\{\mathrm{e}^{\Lambda}_{\mathrm{left}(v)} + \mathrm{e}_{\mathrm{right}(v)},\ \mathrm{e}_{\mathrm{left}(v)} + \mathrm{e}^{\Lambda}_{\mathrm{right}(v)}\right\} \tag{7}$$

$$\mathrm{Env}_v = \max\left\{\mathrm{Env}_{\mathrm{left}(v)} + \mathrm{e}_{\mathrm{right}(v)},\ \mathrm{Env}_{\mathrm{right}(v)}\right\} \tag{8}$$

$$\mathrm{Env}^{\Lambda}_v = \max\left\{\mathrm{Env}^{\Lambda}_{\mathrm{left}(v)} + \mathrm{e}_{\mathrm{right}(v)},\ \mathrm{Env}_{\mathrm{left}(v)} + \mathrm{e}^{\Lambda}_{\mathrm{right}(v)},\ \mathrm{Env}^{\Lambda}_{\mathrm{right}(v)}\right\} \tag{9}$$

Proof. First notice that formulas (6) and (8) are the same as formulas (4) and (5) in Proposition 1. Addition of new leaves representing Λ into the tree cannot invalidate these formulas because for these leaves v we have $\mathrm{e}_v = 0$ and $\mathrm{Env}_v = -\infty$. Therefore formulas (6) and (8) hold by Proposition 1.

Formula (7) is simple to prove. It is enough to realize that the difference between computation of e_v by (6) and computation of e^{Λ}_v is that it is allowed to use one of the activities $i \in \Lambda$. This activity i can be either in the left subtree of v (and in this case we can use $\mathrm{e}^{\Lambda}_{\mathrm{left}(v)}$ instead of $\mathrm{e}_{\mathrm{left}(v)}$) or in the right subtree of v (and we can use $\mathrm{e}^{\Lambda}_{\mathrm{right}(v)}$ instead of $\mathrm{e}_{\mathrm{right}(v)}$). Putting this together we transform formula (6) into (7).

It remains to prove formula (9). Again the difference between computation of Env_v and Env^{Λ}_v is that it is allowed to use one of the activities $i \in \Lambda$. This activity can be either in the left subtree of v (and in this case we can use $\mathrm{Env}^{\Lambda}_{\mathrm{left}(v)}$ instead of $\mathrm{Env}_{\mathrm{left}(v)}$) or in the right subtree of v (and we can use $\mathrm{Env}^{\Lambda}_{\mathrm{right}(v)}$ instead of

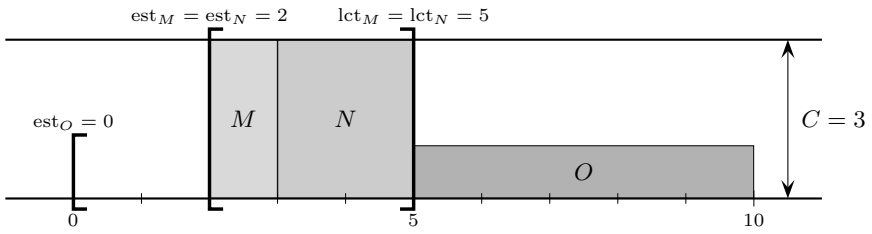

Fig. 4. An example: $\{M, N\} \lessdot O$ but the rule (EF1) is not able to detect it.

$\mathrm{Env}^{\Lambda}_{\mathrm{right}(v)}$ or $\mathrm{e}^{\Lambda}_{\mathrm{right}(v)}$ instead of $\mathrm{e}_{\mathrm{right}(v)}$ but not both). This way we transform formula (8) into (9). □

Thanks to these recursive formulas it is possible to recompute internal values within standard operations with balanced binary trees without changing their time complexity. Therefore lines 9, 11 and 12 of Algorithm 1.1 has time complexity $\mathcal{O}(\log n)$ and line 6 has time complexity $\mathcal{O}(1)$. To prove that time complexity of the whole Algorithm 1.1 is $\mathcal{O}(n \log n)$ it remains to show that time complexity of line 7 is also $\mathcal{O}(\log n)$.

The activity $i \in \Lambda$ responsible for $\mathrm{Env}(\Theta, \Lambda)$ can be found by following a path from the root of the tree to the responsible leaf. In each internal node we can recognize in $\mathcal{O}(1)$ whether the responsible activity is in the left or right subtree by analyzing which part of the formulas (9) or (7) was used in the given node:

$$\mathrm{responsible}_{\mathrm{e}^{\Lambda}}(v) = \begin{cases} \mathrm{responsible}_{\mathrm{e}^{\Lambda}}(\mathrm{left}(v)) & \text{if } \mathrm{e}^{\Lambda}(v) = \mathrm{e}^{\Lambda}_{\mathrm{left}(v)} + \mathrm{e}_{\mathrm{right}(v)} \\ \mathrm{responsible}_{\mathrm{e}^{\Lambda}}(\mathrm{right}(v)) & \text{if } \mathrm{e}^{\Lambda}(v) = \mathrm{e}_{\mathrm{left}(v)} + \mathrm{e}^{\Lambda}_{\mathrm{right}(v)} \end{cases}$$

$$\mathrm{responsible}_{\mathrm{Env}^{\Lambda}}(v) = \begin{cases} \mathrm{responsible}_{\mathrm{Env}^{\Lambda}}(\mathrm{right}(v)) & \text{if } \mathrm{Env}^{\Lambda}(v) = \mathrm{Env}^{\Lambda}_{\mathrm{right}(v)} \\ \mathrm{responsible}_{\mathrm{e}^{\Lambda}}(\mathrm{right}(v)) & \text{if } \mathrm{Env}^{\Lambda}(v) = \mathrm{Env}_{\mathrm{left}(v)} + \mathrm{e}^{\Lambda}_{\mathrm{right}(v)} \\ \mathrm{responsible}_{\mathrm{Env}^{\Lambda}}(\mathrm{left}(v)) & \text{if } \mathrm{Env}^{\Lambda}(v) = \mathrm{Env}^{\Lambda}_{\mathrm{left}(v)} + \mathrm{e}_{\mathrm{right}(v)} \end{cases}$$

We start the search in the root node r looking for $\mathrm{responsible}_{\mathrm{Env}^{\Lambda}}(r)$ and continue down the tree using the formulas above (and possibly switching from $\mathrm{responsible}_{\mathrm{Env}^{\Lambda}}(v)$ to $\mathrm{responsible}_{\mathrm{e}^{\Lambda}}(v)$ on the path) until we reach a leaf.

6.2 Improving Detection

Consider the example on Figure 4. In this example we can see that in every solution $\mathrm{end}(M) \leq \mathrm{end}(O)$ because the maximum possible value for $\mathrm{end}(M)$ is $\mathrm{lct}_M = 5$ and the minimum possible value for $\mathrm{end}(O)$ is $\mathrm{est}_O + \mathrm{p}_O = 5$. Therefore $M \lessdot O$. Similarly $N \lessdot O$. However Edge Finding rule (EF1) is not able to detect that $\{M, N\} \lessdot O$ and we miss the update of est_O from 0 to 5. It is a similar situation to Detectable Precedences for unary resource described in [7].

The idea is to improve the propagation by improving the knowledge of the relation $\lessdot$ stored in the array `prec`:

$$\texttt{prec}[i] := \max\{\texttt{prec}[i],\ \mathrm{est}_i + \mathrm{p}_i\}$$

It takes time $\mathcal{O}(n)$ to update all `prec[i]` according to the formula above.

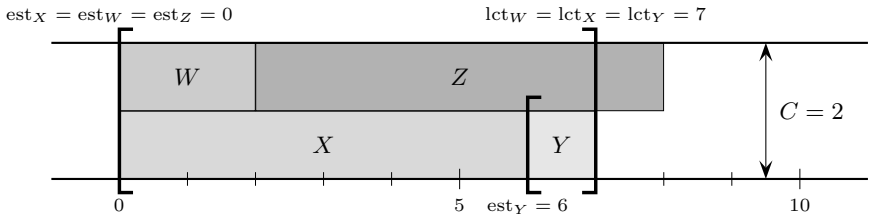

Fig. 5. An example: est_Z can be updated from 0 to 2.

7 Time Bound Adjustment

Let us return again to the example on Figure 1. The algorithm presented in the previous chapter detected that $\{A, B, C\} \lessdot D$. We will try to use this knowledge to update est_D. Notice that activity A is actually not important for the update (but it was important in the previous phase to realize that $\{A, B, C\} \lessdot D$), it is a set $\Omega = \{B, C\} \subset \Theta$ which determines new est_D. With this set Ω we can compute new value of est_D denoted as est'_D:

$$\mathrm{est}'_D = \mathrm{est}_\Omega + \left\lceil \frac{\mathrm{e}_\Omega - (C - \mathrm{c}_D)(\mathrm{lct}_\Omega - \mathrm{est}_\Omega)}{c_D} \right\rceil = 2 + \left\lceil \frac{7 - (3-2)(5-2)}{2} \right\rceil = 4$$

However when $\mathrm{LCut}(T, j) \lessdot i$ we cannot use just any subset $\Omega \subseteq \mathrm{LCut}(T, j)$ to compute update of est_i as we did for est_D above. Consider the example on Figure 5. Here $\{W, X, Y\} \lessdot Z$ but we cannot use $\Omega = \{Y\}$ because the result would be invalid:

$$\mathrm{est}_\Omega + \left\lceil \frac{\mathrm{e}_\Omega - (C - \mathrm{c}_Z)(\mathrm{lct}_\Omega - \mathrm{est}_\Omega)}{c_Z} \right\rceil = 6 + \left\lceil \frac{1 - (2-1)(7-6)}{1} \right\rceil = 6$$

The valid update would be to set est_Z to 2, not to 6. The reason that we cannot use $\Omega = \{Y\}$ for update of est_Z is that there is not enough energy in $\Omega = \{Y\}$ to be in potential conflict with Z.

Let us generalize the idea demonstrated on these examples. When $\mathrm{LCut}(T, j) \lessdot i$ then we want to update est_i the following way:

$$\mathrm{LCut}(T, j) \lessdot i \quad \Rightarrow \quad \mathrm{est}'_i := \max\left\{\mathrm{update}\left(j, \mathrm{c}_i\right),\ \mathrm{est}_i\right\} \qquad \text{(EF2)}$$

where:

$$\mathrm{update}(j, c) = \max_{\substack{\Omega \subseteq \mathrm{LCut}(T,j) \\ \mathrm{e}_\Omega > (C-c)(\mathrm{lct}_\Omega - \mathrm{est}_\Omega)}} \left\{ \mathrm{est}_\Omega + \left\lceil \frac{\mathrm{e}_\Omega - (C - c)\left(\mathrm{lct}_\Omega - \mathrm{est}_\Omega\right)}{c} \right\rceil \right\}$$

The condition $\mathrm{e}_\Omega > (C - c)(\mathrm{lct}_\Omega - \mathrm{est}_\Omega)$ makes sure that we do not make any invalid update as described above.

In the following we will describe how to compute values $\mathrm{update}(j, c)$. When all values $\mathrm{update}(j, c)$ are computed then update of est_i using array `prec` and formula (EF2) is trivial.

Let's assume for simplicity that there are no duplicates in the set $\{\mathrm{lct}_j,\ j \in T\}$. Therefore if we sort activities T by increasing lct_j in a sequence $j_1, j_2, \ldots, j_n$ we get:

$$\mathrm{LCut}(T, j_1) \subsetneq \mathrm{LCut}(T, j_2) \subsetneq \cdots \subsetneq \mathrm{LCut}(T, j_n)$$

So when we compute value $\mathrm{update}(j_l, c)$ we do not have to iterate again on all possible subsets $\Omega \subseteq \mathrm{LCut}(T, j_l)$, we can use the fact that we already considered part of them in the computation of $\mathrm{update}(j_{l-1}, c)$. I.e. in the outermost cycle of the algorithm we iterate over all $c \in \{c_m,\ m \in T\}$ and in the inner cycle we iterate over all $j_l \in T$ and compute:

$$\mathrm{update}(j_l, c) = \begin{cases} \mathrm{diff}\,(j_1, c) & \text{when } l = 1 \\ \max\{\mathrm{update}\,(j_{l-1}, c)\,,\ \mathrm{diff}\,(j_l, c)\} & \text{when } l > 1 \end{cases} \tag{10}$$

where:

$$\mathrm{diff}(j, c) = \max_{\substack{\Omega \subseteq \mathrm{LCut}(T,j) \\ \mathrm{e}_\Omega > (C-c)(\mathrm{lct}_j - \mathrm{est}_\Omega)}} \left\{ \mathrm{est}_\Omega + \left\lceil \frac{\mathrm{e}_\Omega - (C-c)(\mathrm{lct}_j - \mathrm{est}_\Omega)}{c} \right\rceil \right\} \tag{11}$$

So in the computation of $\mathrm{diff}(j, c)$ we "pretend" that all sets $\Omega \subseteq \mathrm{LCut}(T, j)$ has $\mathrm{lct}_\Omega = \mathrm{lct}_j$. This is not true, there may be sets $\Omega \subsetneq \mathrm{LCut}(T, j)$ such that $\mathrm{lct}_\Omega < \mathrm{lct}_j$. However these sets are correctly considered during computation of $\mathrm{diff}(j', c)$ such that $\mathrm{lct}_{j'} = \mathrm{lct}_\Omega$.

Let's define[2] function $\mathrm{maxest}(j, c)$ as:

$$\mathrm{maxest}(j, c) = \max\{\mathrm{est}_\Omega,\ \Omega \subseteq \mathrm{LCut}\,(T, j)\ \&\ \mathrm{e}_\Omega > (C-c)(\mathrm{lct}_j - \mathrm{est}_\Omega)\}$$

Notice that for a particular set Ω_m which defines $\mathrm{maxest}(j, c)$, i.e. $\mathrm{est}_{\Omega_m} = \mathrm{maxest}(j, c)$, we have:

$$\mathrm{est}_{\Omega_m} + \left\lceil \frac{\mathrm{e}_{\Omega_m} - (C-c)(\mathrm{lct}_j - \mathrm{est}_\Omega)}{c} \right\rceil > \mathrm{est}_{\Omega_m} = \mathrm{maxest}(j, c)$$

and therefore $\mathrm{diff}(j, c) > \mathrm{maxest}(j, c)$. Now we will show that:

$$\mathrm{diff}(j, c) = \max_{\substack{\Omega \subseteq \mathrm{LCut}(T,j) \\ \mathrm{est}_\Omega \leq \mathrm{maxest}(j,c)}} \left\{ \mathrm{est}_\Omega + \left\lceil \frac{\mathrm{e}_\Omega - (C-c)(\mathrm{lct}_j - \mathrm{est}_\Omega)}{c} \right\rceil \right\} \tag{12}$$

The reason follows. The original condition was more restrictive than the new one: in (12) we iterate over more sets Ω than in (11). However for every additional set Ω we have $\mathrm{est}_\Omega \leq \mathrm{maxest}(j, c)$ and $\mathrm{e}_\Omega \leq (C-c)(\mathrm{lct}_j - \mathrm{est}_\Omega)$ therefore:

$$\mathrm{est}_\Omega + \left\lceil \frac{\mathrm{e}_\Omega - (C-c)(\mathrm{lct}_j - \mathrm{est}_\Omega)}{c} \right\rceil \leq \mathrm{est}_\Omega \leq \mathrm{maxest}(j, c)$$

And we already know that $\mathrm{diff}(j, c) > \mathrm{maxest}(j, c)$. Therefore newly added sets cannot influence the resulting maximum value in formula (12).

[2] Thanks to Luc Mercier and Joseph Scott for correcting min/max error in the paper.

Formula (12) is algebraically equivalent to:

$$\mathrm{diff}(j,c) = \left\lceil \frac{\mathrm{Env}(j,c) - (C-c)\,\mathrm{lct}_j}{c} \right\rceil \tag{13}$$

where:

$$\mathrm{Env}(j,c) = \max_{\substack{\Omega \subseteq \mathrm{LCut}(T,j) \\ \mathrm{e}_\Omega > (C-c)(\mathrm{lct}_j - \mathrm{est}_\Omega)}} \{C\,\mathrm{est}_\Omega + \mathrm{e}_\Omega\} \tag{14}$$

We can split each set Ω by $\mathrm{maxest}(j,c)$ into two parts:

$$\Omega_1 = \{l,\ l \in \Omega\ \&\ \mathrm{est}_l \leq \mathrm{maxest}\,(j,c)\}$$
$$\Omega_2 = \{l,\ l \in \Omega\ \&\ \mathrm{est}_l > \mathrm{maxest}\,(j,c)\}$$

And then $C\,\mathrm{est}_\Omega + \mathrm{e}_\Omega = C\,\mathrm{est}_{\Omega_1} + \mathrm{e}_{\Omega_1} + \mathrm{e}_{\Omega_2}$. Let's apply this idea on formula (14). We define:

$$\alpha(j,c) = \{l,\ l \in \mathrm{LCut}\,(T,j)\ \&\ \mathrm{est}_l \leq \mathrm{maxest}\,(j,c)\}$$
$$\beta(j,c) = \{l,\ l \in \mathrm{LCut}\,(T,j)\ \&\ \mathrm{est}_l > \mathrm{maxest}\,(j,c)\}$$

And (14) is equivalent to:

$$\begin{aligned}\mathrm{Env}(j,c) &= \max_{\substack{\Omega_1 \subseteq \alpha(j,c) \\ \Omega_2 \subseteq \beta(j,c)}} \{C\,\mathrm{est}_{\Omega_1} + \mathrm{e}_{\Omega_1} + \mathrm{e}_{\Omega_2}\} = \\ &= \mathrm{e}_{\beta(j,c)} + \mathrm{Env}\,(\alpha\,(j,c))\end{aligned} \tag{15}$$

We can compute $\mathrm{Env}\,(\alpha\,(j,c))$ by building Θ-tree for the set $\alpha(j,c)$ as shown in Proposition 1. However it is more suitable for the algorithm to build Θ-tree for the whole set $\mathrm{LCut}(T,j)$ and cut it into two parts just before the computation of $\mathrm{Env}\,(\alpha\,(j,c))$. The *cut* operation splits the tree into two trees, it is done in such a way that all activities $l \in \mathrm{LCut}(T,j)$ such that $\mathrm{est}_l \leq \mathrm{maxest}(j,c)$ go to the left part while the others go into the right part. See Figure 6 for an example. The cut operation has time complexity $\mathcal{O}(\log n)$ and it splits the set $\mathrm{LCut}(T,j)$ into sets $\alpha(j,c)$ and $\beta(j,c)$. The value $\mathrm{Env}\,(\alpha\,(j,c))$ can be found in the root node of the Θ-tree for $\alpha(j,c)$, and $\mathrm{e}_{\beta(j,c)}$ can be found in the root node of the Θ-tree for $\beta(j,c)$.

It remains to show how to compute $\mathrm{maxest}(j,c)$. The value $\mathrm{maxest}(j,c)$ was defined as:

$$\mathrm{maxest}(j,c) = \min\{\mathrm{est}_\Omega,\ \Omega \subseteq \mathrm{LCut}\,(T,j)\ \&\ \mathrm{e}_\Omega > (C-c)(\mathrm{lct}_j - \mathrm{est}_\Omega)\}$$

The condition $\mathrm{e}_\Omega > (C-c)(\mathrm{lct}_j - \mathrm{est}_\Omega)$ is algebraically equivalent to:

$$(C-c)\,\mathrm{est}_\Omega + \mathrm{e}_\Omega > (C-c)\,\mathrm{lct}_j \tag{16}$$

Notice that the left part of this inequality is very similar to the computation of energy envelope, just C is replaced by $(C-c)$. Let us define a new variant of energy envelope Env^c:

$$\mathrm{Env}^c(\Theta) = \max_{\Omega \subseteq \Theta} \{(C-c)\,\mathrm{est}_\Omega + \mathrm{e}_\Omega\}$$

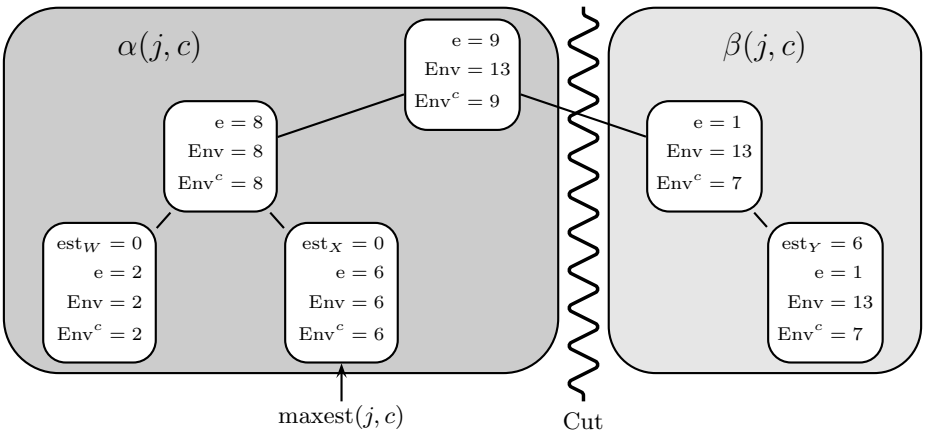

Fig. 6. Example: computation of $\text{Env}(j,c)$ for $c = 1$ and $j = Y$ from example on Figure 5. Therefore $\text{LCut}(T,j) = \{W,X,Y\}$. Situation just before the cut.

The computation of Env^c can be done again using Θ-tree by Proposition 1. We can compute Env and Env^c in the same Θ-tree as shown on Figure 6. Now we can see that because of condition (16) it must hold[3]:

$$\text{Env}^c\left(\beta\left(j,c\right)\right) \leq \left(C-c\right)\text{lct}_j$$

but if we would include activities l with $\text{est}_l = \text{maxest}(j,c)$ into the right tree (in other words if we would do the cut more on the left) then this condition would not hold. That allows to find a leaf l with $\text{est}_l = \text{maxest}(j,c)$ by following a path from the root the leaf as shown in Algorithm 1.2. Using this procedure we can compute all values $\text{update}(j,c)$ by Algorithm[4] 1.3 with time complexity $\mathcal{O}(kn\log n)$. Note that once $\text{update}(j,c)$ is computed we can trivially update values est_i using (EF2).

8 Relation with standard Edge Finding

We will show that the algorithm described in the paper does not miss any update done by Edge Finding algorithm described in [5]. It is enough to prove that the original Edge Finding propagation rules are subsumed by the new rules (EF1) and (EF2).

The traditional Edge Finding rule is:

$$\forall i \in T,\ \forall \Theta \subseteq T\backslash\Theta:\ \ C\left(\text{lct}_\Theta - \text{est}_{\Theta\cup\{i\}}\right) < \text{e}_{\Theta\cup\{i\}} \ \Rightarrow\ \text{est}_i := \max\left(\text{est}_i, \text{newest}_i\right)$$

where:

$$\text{newest}_i = \max_{\substack{\Omega\subseteq\Theta \\ \text{e}_\Omega > (C-c)(\text{lct}_\Omega - \text{est}_\Omega)}} \left\{\text{est}_\Omega + \left\lceil \frac{\text{e}_\Omega - (C-c)\left(\text{lct}_\Omega - \text{est}_\Omega\right)}{c_i} \right\rceil\right\} \quad (17)$$

[3] Thanks to Albert Oliveras Llunell for correcting a typo in the original paper.
[4] Thanks to Joseph Scott for correcting a typo on line 9.

Algorithm 1.2. Computation of maxest(j, c) using Θ-tree for LCut(T, j)

```
1  v := root;
2  E := 0;
3  while v is not a leaf node do begin
4    if Env^c(right(v)) + E > (C − c) lct_j then
5      v := right(v);
6    else begin
7      E := E + e_right(v);
8      v := left(v);
9    end;
10 end;
11 l := activity represented by leaf v;
12 return est_l;
```

Algorithm 1.3. Computation of all update(j, c)

```
1  for c ∈ {c_m, m ∈ T} do begin
2    Θ := ∅;
3    upd := −∞;
4    for j ∈ T in non-decreasing order by lct_j do begin
5      Θ := Θ ∪ {j};
6      maxest := maxest(j, c);                  // see Algorithm 1.2
7      (α, β) := Cut(Θ, maxest);
8      Env(j, c) := e(β) + Env(α);              // see (15)
9      diff := ⌈(Env(j,c) − (C − c) lct_j) / c⌉;   // see (13)
10     upd := max(upd, diff);                   // see (10)
11     update(j, c) := upd;
12     Θ := join(α, β);
13   end;
14 end;
```

Let's consider an activity i and sets Θ and Ω which achieves the best update by the rule above. Then we can define j to be an activity from Θ such that $\mathrm{lct}_j = \mathrm{lct}_\Theta$. And because $\Theta \subseteq \mathrm{LCut}(T, j)$ we can see that the rule (EF1) holds for i and j. And because $\Omega \subseteq \Theta \subseteq \mathrm{LCut}(T, j)$ the update by the rule (EF2) must be at least the same as by the original rule (17).

9 Optional Activities

Optional activity is an activity which may or may not be present in the resulting schedule [1]. Optional activities makes modeling of certain types of problems much easier (for example dealing with alternatives) and it also allows the CP engine to propagate better. Therefore it is very important that Edge Finding algorithm can handle optional activities.

To handle optional activities we can use the same idea as suggested in [4]: instead of changing the algorithm we can just change its input data. If an activity j is optional, we set for the algorithm $\mathrm{lct}_j = \infty$ regardless the real value of lct_j. This way the algorithm can never conclude that $j \ll i$ for any activity i because from the point of view of the algorithm the activity j can be always scheduled later than i. Therefore optional activities will be influenced by non-optional ones, but non-optional activities will not be influenced by optional ones.

Note that propagation for optional activities could be further improved as suggested for unary resource in [6]. However it would probably result in increase of time complexity of the algorithm.

10 Experimental Results

Speed of the presented algorithm was tested against incomplete algorithm [2] by measuring time needed for initial propagation. These tests was done on cumulative job-shop instances with resources of capacity 2 (note that in this case $k = 1$). For $n = 20$ activities on resource the presented algorithm is on average faster by factor 1.34, for $n = 30$ it is faster by factor 1.60, for $n = 40$ by 1.99, for $n = 60$ by 2.68, for $n = 100$ by 4.15 and for $n = 200$ by factor 7.35.

11 Conclusions

This paper presents a new Edge Finding algorithm for discrete capacity resources. The new algorithm is stronger than the state-of-the-art algorithm [5], it is faster (in term of time complexity) and it can handle optional activities. The algorithm is successfully used by CP-Optimizer [1] starting from version 2.0.

References

1. : IBM ILOG CP Optimizer `http://www.ilog.com/products/cpoptimizer/`.
2. Philippe Baptiste, C.L.P., Nuijten, W.: Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems. Kluwer Academic Publishers (2001)
3. Schutt, A., Wolf, A., Schrader, G.: Not-first and not-last detection for cumulative scheduling in $O(n^3 \log n)$. In: 16th International Conference on Applications of Declarative Programming and Knowledge Management, INAP 2005, Springer (2005) 66–80
4. Vilím, P.: Max energy filtering algorithm for discrete cumulative resources. In Willem-Jan van Hoeve, J.N.H., ed.: Proceedings of CP-AI-OR 2009. Volume 5547 of Lecture Notes in Computer Science., Springer-Verlag (2009) 294–308
5. Mercier, L., Hentenryck, P.V.: Edge finding for cumulative scheduling. Informs Journal of Computing **20**(1) (2008) 143–153
6. Kuhnert, S.: Efficient edge-finding on unary resources with optional activities. In: Proceedings of INAP 2007 and WLP 2007. (2007)
7. Vilím, P.: Global Constraints in Scheduling. PhD thesis, Charles University in Prague, Faculty of Mathematics and Physics, Department of Theoretical Computer Science and Mathematical Logic (2007)