Max Energy Filtering Algorithm for Discrete Cumulative Resources

Petr Vilím

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

Abstract. In scheduling using constraint programming we usually reason only about possible start times and end times of activities and remove those which are recognized as unfeasible. However often in practice there are more variables in play: variable durations of activities and variable resource capacity requirements. This paper presents a new algorithm for filtering maximum durations and maximum capacity requirements for discrete cumulative resources. It is also able to handle optional interval variables introduced in IBM ILOG CP Optimizer 2.0. Time complexity of the algorithm is $\mathcal{O}(n \log n)$. The algorithm is based on never published algorithm by Wim Nuijten and a on slightly modified e-feasibility checking algorithm by Armin Wolf and Gunnar Schrader. The later algorithm is also described in the paper.

Keywords: Constraint Programming, Scheduling, Discrete Cumulative Resource, Propagation.

1 Introduction

Nowadays, constraint based scheduling engines like IBM ILOG CP Optimizer [1] allows to describe and solve very complex scheduling problems involving a variety of different constraints. This paper is focused on one of them – discrete cumulative resource for the case when durations and/or individual capacity requirements are not fixed. Traditionally we reason only about minimum start times and maximum end times using algorithms like Edge Finding [2], Not-First/Not-Last [5] or Energetic Reasoning [3]. This paper provides an algorithm for filtering of maximum activity durations and maximum capacity requirements.

The algorithm is not completely new. Although it was never published, Wim Nuijten implemented a similar algorithm for ILOG Scheduler several years ago. The old version of the algorithm has time complexity $\mathcal{O}(n^2)$, this paper presents a faster version with time complexity $\mathcal{O}(n \log n)$.

To demonstrate the problem on a simple example, lets consider the following subproblem: there is a pool of 10 workers (i.e., a discrete capacity resource with maximum capacity $C = 10$) who perform different tasks. Among these tasks there is a task to produce one particular product P. How many units of the product P is produced depends on how many workers are assigned to the tasks

W.-J. van Hoeve and J.N. Hooker (Eds.): CPAIOR 2009, LNCS 5547, pp. 294–308, 2009.
© Springer-Verlag Berlin Heidelberg 2009

(i.e., how much capacity of the resource is used) and for how long (i.e., what is the duration of the task):

$$\text{nbP} = \text{workers} \times \text{duration}$$

If we do not produce at least 500 units of product P then we will have to buy the rest for the following cost (deduced from an initial budget):

$$\text{cost} = \max(0, 500 - \text{nbP}) \times 1\$$$

In this example, the budget and production of product P are tightly connected:

1. If we see that no more than 200$ can be invested into the purchase of product P (because the rest of the budget is needed for other things) then we need to allocate workers to produce at least 300 units of product P.
2. On the other hand if we see that there is no way to produce more than 100 units of product P (because the workers are needed for other tasks) we can immediately allocate 400$ from the budget to buy remaining products P. This is a critical propagation especially if the budget is short.

Both propagations above are very important for speeding up the search by better pruning the search tree. However for the propagation 2 it is necessary to be able to compute the maximum possible production of product P. And this is the topic of the paper.

The algorithm presented in the paper is also useful if there are optional activities – activities which may or may not be present in the solution (for example alternatives between several resources). In this case the algorithm can detect that there is no way to process an optional activity and therefore it cannot be present in the solution (and, in case of an alternative, another alternative must be chosen), see [4, 1].

2 Notation

Let us formalize the problem. There is a set T of $n = |T|$ non-preemptive non-optional activities. For the first part of the paper we assume that none of the activities in T is optional, that is, all activities in T are necessarily present in the solution. After we present Max Energy algorithm for non-optional activities we will show how to use it for optional activities.

Each activity $i \in T$ is described by the following attributes:

- the earliest possible starting time $\text{est}_i \in \mathbb{N}$,
- the latest possible completion time $\text{lct}_i \in \mathbb{N}$,
- the minimum processing time (duration) $\text{p}_i \in \mathbb{N}$,
- the maximum processing time (duration) $\overline{\text{p}}_i \in \mathbb{N}$.

Moreover, each activity $i \in T$ consumes during its processing some capacity of a resource. The capacity consumption during the whole processing of the activity is constant, however it may not be known in advance. In this case there is a range of possible capacity consumption:

- the minimum required capacity $c_i \in \mathbb{N}$,
- the maximum required capacity $\overline{c}_i \in \mathbb{N}$.

The resource can process several activities at the same time, however at any time the total used capacity cannot exceed the maximum resource capacity C. For an example see Figure 1.

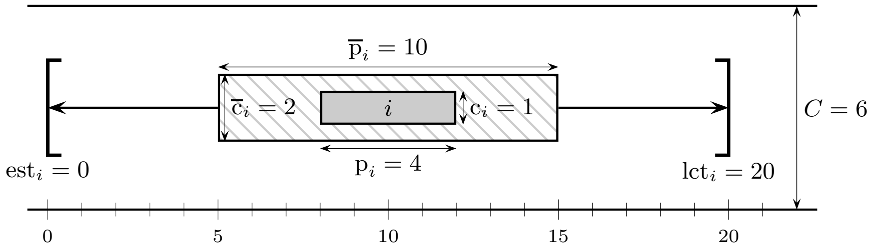

Fig. 1. An example of an activity i on a resource with capacity $C = 6$

Another way to characterize an activity is its energy. Informally, energy of an activity is:

$$\text{energy} = \text{processingTime} \times \text{capacity}$$

Because processing time and/or required capacity may be unbound, we characterize the energy of a task i by two numbers: minimum energy $e_i = c_i\, p_i$ and maximum energy $\overline{e}_i = \overline{c}_i \overline{p}_i$. The presented algorithm modifies maximum energy $\overline{e}_i$ and this way also maximum capacity $\overline{c}_i$ and maximum processing time $\overline{p}_i$:

$$\overline{c}_i := \min \left\{ \lfloor \overline{e}_i / p_i \rfloor , \; \overline{c}_i \right\} \tag{1}$$

$$\overline{p}_i := \min \left\{ \lfloor \overline{e}_i / c_i \rfloor , \; \overline{p}_i \right\} \tag{2}$$

2.1 Earliest Completion Time, Energy Envelope

For the following algorithms we need a way to quickly estimate the earliest completion time of any set of activities $\Theta \subseteq T$. If Θ contains only one activity i then the computation of the earliest completion time is simple:

$$\text{ect}_i = \text{est}_i + p_i$$

However in the general case it is much more complicated. Therefore we are looking for a good lower bound, traditionally defined as:

$$\text{preEct}(\Theta) = \text{est}_\Theta + \left\lceil \frac{e_\Theta}{C} \right\rceil$$

where:

$$\text{est}_\Theta = \min_{i \in \Theta} \{\text{est}_i\}$$

$$e_\Theta = \sum_{i \in \Theta} e_i$$

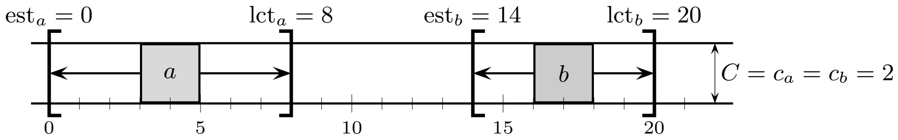

Fig. 2. An example: preEct $(\{a, b\}) = 4$ even though $\{a, b\}$ cannot end before $\mathrm{ect}_b = 16$

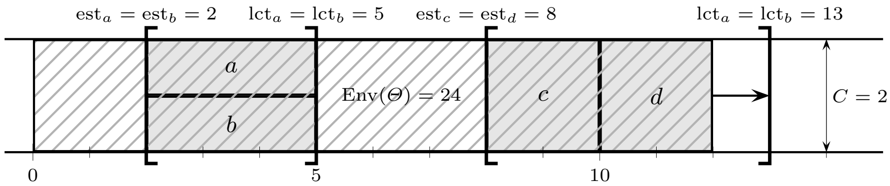

Fig. 3. Example of a set $\Theta = \{a, b, c, d\}$ with earliest completion time $\mathrm{Ect}(\Theta) = 12$ and energy envelope $\mathrm{Env}(\Theta) = 24$. Maximum envelope is achieved by the set $\Omega = \{c, d\}$. Energy envelope is depicted by gray lines.

Note that preEct is only a lower bound. For example:

(i) If we closely inspect all subsets $\Omega \subseteq \Theta$ we can achieve better estimation of earliest completion time of the set Θ. For an example Figure 2.
(ii) Since we take into account only total energy of the set Θ, we assume that all activities in Θ are fully "elastic". For example for $\Theta = \{i\}$ from Figure 1 $\mathrm{preEct}(\Theta) = 1$ even though activity i cannot end before $\mathrm{ect}_i = 4$. [1]

In this paper we will address only issue (i) by defining better estimation of earliest completion time:

$$\mathrm{Ect}(\Theta) = \max_{\Omega \subseteq \Theta} \{\mathrm{preEct}(\Omega)\}$$

What is algebraically equivalent to:

$$\mathrm{Ect}(\Theta) = \max_{\Omega \subseteq \Theta} \left\{ \mathrm{est}_\Omega + \left\lceil \frac{\mathrm{e}_\Omega}{C} \right\rceil \right\} = \left\lceil \frac{\max_{\Omega \subseteq \Theta}\{C \,\mathrm{est}_\Omega + \mathrm{e}_\Omega\}}{C} \right\rceil$$

Lets call the numerator of the last fraction *energy envelope* of the set Θ:

$$\mathrm{Env}(\Theta) = \max_{\Omega \subseteq \Theta} \{C \,\mathrm{est}_\Omega + \mathrm{e}_\Omega\} \tag{3}$$

Hence:

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

For an example of $\mathrm{Ect}(\Theta)$ and $\mathrm{Env}(\Theta)$ see Figure 3.

The reason we defined energy envelope is that it is simpler to use in the algorithms than the earliest completion time.

[1] That is also the reason why earliest start time of an activity i is denoted by lower-case letters ect_i but earliest completion time of a set of activities Θ is denoted by $\mathrm{Ect}(\Theta)$ with capital E.

3 Overload, E-Feasibility

This section provides a variation of the e-feasibility checking algorithm by Armin Wolf and Gunnar Schrader [7]. This algorithm is the basis of the Max Energy algorithm presented later in this paper.

Traditionally, we define an *overload* as a situation when a subset of activities $\Omega \subseteq T$ requires more resource energy than what is available between earliest possible start and latest possible end time of the set Ω (see for example [3]). If there is overload then no solution exists:

$$\forall \Omega \subseteq T : \quad (\mathrm{e}_\Omega > C(\mathrm{lct}_\Omega - \mathrm{est}_\Omega) \quad \Rightarrow \quad \text{fail}) \qquad \text{(OL)}$$

where:

$$\mathrm{lct}_\Omega = \max\{\mathrm{lct}_i,\ i \in \Omega\}$$

If there is no overload then we say that the problem is *e-feasible.*

It would take too much time to check all subsets $\Omega \subseteq T$. Fortunately there is a faster way:

Proposition 1. *The problem is e-feasible if and only if*

$$\forall j \in T : \quad \mathrm{Env}\left(\mathrm{LCut}\left(T, j\right)\right) \leq C\,\mathrm{lct}_j$$

where $\mathrm{LCut}(T, j)$ *is a* left cut of T by activity j*:*

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

Proof. We will prove the equivalence by proving both implications:

1. If rule (OL) detects overload then there is a set Ω such that $C\,\mathrm{lct}_\Omega < C\,\mathrm{est}_\Omega + \mathrm{e}_\Omega$. In this case we define $j \in \Omega$ to be activity from set Ω such that $\mathrm{lct}_j = \mathrm{lct}_\Omega$ (if there are more activities with this property, we can choose arbitrarily). Thanks to the definition of j it holds that $\Omega \subseteq \mathrm{LCut}(T, j)$ and therefore:

$$C\,\mathrm{lct}_j = C\,\mathrm{lct}_\Omega < C\,\mathrm{est}_\Omega + \mathrm{e}_\Omega \overset{(3)}{\leq} \mathrm{Env}\left(\mathrm{LCut}\left(T, j\right)\right)$$

   Therefore the second rule also detects overload.
2. If $\mathrm{Env}\left(\mathrm{LCut}\left(T, j\right)\right) > C\,\mathrm{lct}_j$ then by (3) there is a set $\Omega \subseteq \mathrm{LCut}(T, j)$ such that $C\,\mathrm{est}_\Omega + \mathrm{e}_\Omega = \mathrm{Env}\left(\mathrm{LCut}\left(T, j\right)\right)$. And for this set Ω:

$$C\,\mathrm{lct}_\Omega \leq C\,\mathrm{lct}_j < \mathrm{Env}\left(\mathrm{LCut}\left(T, j\right)\right) = C\,\mathrm{est}_\Omega + \mathrm{e}_\Omega$$

   And therefore rule (OL) also detects overload. □

The key idea of the algorithm is to organize set $\mathrm{LCut}(T, j) = \Theta$ in a balanced binary tree, which we call Θ-tree (it is an extension of Θ-tree structure for unary resources described for example in [6]). Activities are represented by leaf nodes[2]

[2] This is the main difference from the algorithm in [7], that algorithm represents activities also in the internal nodes of the tree.

and sorted by est_i from left to right. Each node v of the tree holds the following values:

$$\text{e}_v = \text{e}_{\text{Leaves}(v)} \tag{4}$$
$$\text{Env}_v = \text{Env}\left(\text{Leaves}\left(v\right)\right) \tag{5}$$

Where Leaves(v) is a set of all activities represented by leaves of the subtree rooted in v. Figure 4 shows a Θ-tree from an example from Figure 3. Notice that the energy envelope of the represented set Θ is equivalent to the value Env of the root node.

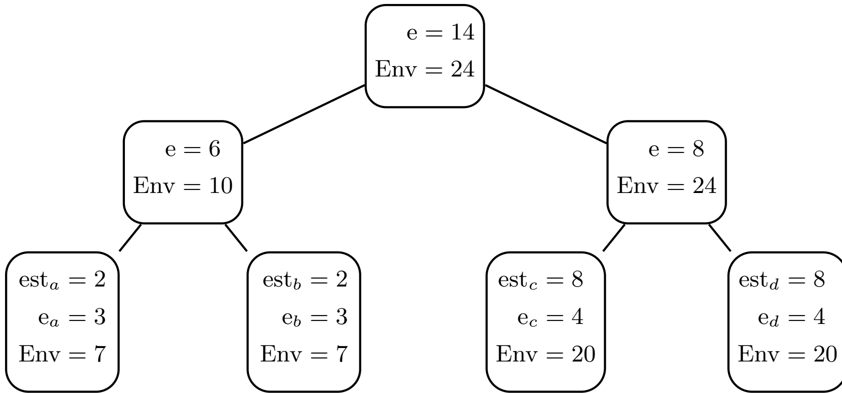

Fig. 4. An example of a Θ-tree for $\Theta = \{a, b, c, d\}$ from Figure 3

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

$$\text{e}_v = \text{e}_i$$
$$\text{Env}_v = \text{Env}\left(\{i\}\right) = C\,\text{est}_i + \text{e}_i$$

For internal nodes v these values can be computed recursively from their children nodes left(v) and right(v):

Proposition 2. *For an internal node v, values e_v and Env_v can be computed by the following recursive formula:*

$$\text{e}_v = \text{e}_{\text{left}(v)} + \text{e}_{\text{right}(v)} \tag{6}$$
$$\text{Env}_v = \max\left\{\text{Env}_{\text{left}(v)} + \text{e}_{\text{right}(v)},\ \text{Env}_{\text{right}(v)}\right\} \tag{7}$$

Proof. Formula (6) is trivial, we will prove only formula (7). From the definition (5), the value Env_v is:

$$\text{Env}_v = \text{Env}\left(\text{Leaves}\left(v\right)\right) = \max\left\{C\,\text{est}_\Omega + \text{e}_\Omega,\ \Omega \subseteq \text{Leaves}(v)\right\}$$

With respect to the node v we will split the sets Ω into the following two categories:

1. $\mathrm{Left}(v) \cap \Omega = \emptyset$, i.e., $\Omega \subseteq \mathrm{Right}(v)$. Clearly:

$$\max\{C\,\mathrm{est}_\Omega + \mathrm{e}_\Omega,\ \Omega \subseteq \mathrm{Right}(v)\} = \mathrm{Env}\left(\mathrm{Right}(v)\right) = \mathrm{Env}_{\mathrm{right}(v)}$$

2. $\mathrm{Left}(v) \cap \Omega \neq \emptyset$. Then $\mathrm{est}_\Omega = \mathrm{est}_{\Omega \cap \mathrm{Left}(v)}$ because leaf nodes are sorted by est_i. Let S be the set of all possible Ω considered in this part of the proof:

$$S = \{\Omega,\ \Omega \subseteq \Theta\ \&\ \Omega \cap \mathrm{Left}(v) \neq \emptyset\}$$

Then:

$$\begin{aligned}
&\max\{C\,\mathrm{est}_\Omega + \mathrm{e}_\Omega,\ \Omega \in S\} = \\
&\quad = \max\left\{C\,\mathrm{est}_{\Omega \cap \mathrm{Left}(v)} + \mathrm{e}_{\Omega \cap \mathrm{Left}(v)} + \mathrm{e}_{\Omega \cap \mathrm{Right}(v)},\ \Omega \in S\right\} = \\
&\quad = \max\left\{C\,\mathrm{est}_{\Omega \cap \mathrm{Left}(v)} + \mathrm{e}_{\Omega \cap \mathrm{Left}(v)},\ \Omega \in S\right\} + \mathrm{e}_{\mathrm{Right}(v)} = \\
&\quad = \mathrm{Env}_{\mathrm{left}(v)} + \mathrm{e}_{\mathrm{right}(v)}
\end{aligned}$$

We used the fact that the maximum is achieved only by such a set Ω for which $\mathrm{Right}(v) \subsetneq \Omega$. We also used the fact that $\Omega \cap \mathrm{Left}(v)$ enumerates all possible subsets of $\mathrm{Left}(v)$ and therefore:

$$\max\left\{C\,\mathrm{est}_{\Omega \cap \mathrm{Left}(v)} + \mathrm{e}_{\Omega \cap \mathrm{Left}(v)},\ \Omega \in S\right\} = \mathrm{Env}_{\mathrm{left}(v)}$$

Combining the results of parts 1 and 2 together we see that formula (7) is correct. □

Thanks to formulas (6) and (7), computation of values e_v and Env_v can be integrated within usual operations with balanced binary trees without changing their time complexity, see Table 1.

Table 1. Worst-case time complexities of operations on Θ-tree

Operation	**Time Complexity**
$\Theta := \emptyset$	$\mathcal{O}(1)$
$\Theta := \Theta \cup \{i\}$	$\mathcal{O}(\log n)$
$\Theta := \Theta \setminus \{i\}$	$\mathcal{O}(\log n)$
$\mathrm{Env}(\Theta)$	$\mathcal{O}(1)$

The idea of the overload checking algorithm follows. We will iterate over all left cuts $\mathrm{LCut}(T, j)$ by non-decreasing lct_j. The cuts will be represented by Θ-tree what allows to quickly recompute $\mathrm{Env}\left(\mathrm{LCut}\left(T, j\right)\right)$ each time when j is changed. For each set $\Theta = \mathrm{LCut}(T, j)$ we check e-feasibility using Proposition 1. The resulting Algorithm 1.1 has worst-case time complexity $\mathcal{O}(n \log n)$.

Algorithm 1.1. Overload Checking in $\mathcal{O}(n \log n)$

```
1  Θ := ∅;
2  for j ∈ T in non-decreasing order of lct_j do begin
3    Θ := Θ ∪ {j};
4    if Env(Θ) > C lct_j then
5      fail; {No solution exists}
6  end;
```

4 Max Energy Propagation

In this section we will extend the algorithm for overload detection to compute maximum energy of each activity $i \in T$. The idea of the propagation is to protect possible overload caused by increase of some energy demand e_i.

Consider for example situation on Figure 3. In this example, minimum required energy of activity c is $\mathrm{e}_c = 4$. Maximum required energy $\overline{\mathrm{e}}_c$ is not depicted on the figure, but lets say that $\overline{\mathrm{e}}_c = 10$. However considering also activity d (which requires at least $\mathrm{e}_d = 4$) the maximum feasible energy for activity c is 6, otherwise there would be an overload for $\Omega = \{c, d\}$. Therefore we can update $\overline{\mathrm{e}}_c := 6$, and according to formula (2) $\overline{\mathrm{p}}_c := 3$. What we just described on the example is the goal of the presented algorithm: for each activity $i \in T$, compute maximum feasible energy $\overline{\mathrm{e}}_i$ such that if e_i is increased above $\overline{\mathrm{e}}_i$ then there will be an overload.

In Proposition 1 we have learned that the resource is e-feasible iff:

$$\forall j \in T : \quad \mathrm{Env}\left(\mathrm{LCut}\left(T, j\right)\right) \leq C\, \mathrm{lct}_j$$

In other words we can assign to each set $\mathrm{LCut}\left(T, j\right)$ a maximum feasible envelope $\overline{\mathrm{Env}}$:

$$\overline{\mathrm{Env}}\left(\mathrm{LCut}\left(T, j\right)\right) := C\, \mathrm{lct}_j \tag{8}$$

The idea is to propagate maximum feasible envelope from the set $\mathrm{LCut}(T, j)$ into all its members and this way find maximum feasible energy of all activities.

Lets have have a look on the Θ-tree representing a particular set $\Theta = \mathrm{LCut}(T, j)$. For overload checking we compute recursively in each node the following values by formulas (6) and (7):

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

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

The idea is to extend the tree by adding two more attributes into each node of the tree:

- maximum feasible energy envelope $\overline{\mathrm{Env}}_v$ of the set $\mathrm{Leaves}(v)$,
- and maximum feasible energy $\overline{\mathrm{e}}_v$ for the set $\mathrm{Leaves}(v)$.

The additional attributes can be also computed recursively, this time from root down to the leaves. It starts at the root node r (see (8)):

$$\overline{\mathrm{Env}}_r := C\,\mathrm{lct}_j \tag{9}$$

$$\overline{\mathrm{e}}_r := \infty \tag{10}$$

The recursive rules to propagate these values down the tree are:

$$\overline{\mathrm{Env}}_{\mathrm{right}(v)} := \overline{\mathrm{Env}}_v \tag{11}$$

$$\overline{\mathrm{Env}}_{\mathrm{left}(v)} := \overline{\mathrm{Env}}_v - \mathrm{e}_{\mathrm{right}(v)} \tag{12}$$

$$\overline{\mathrm{e}}_{\mathrm{right}(v)} := \min\left\{\overline{\mathrm{Env}}_v - \mathrm{Env}_{\mathrm{left}(v)}, \overline{\mathrm{e}}_v - \mathrm{e}_{\mathrm{left}(v)}\right\} \tag{13}$$

$$\overline{\mathrm{e}}_{\mathrm{left}(v)} := \overline{\mathrm{e}}_v - \mathrm{e}_{\mathrm{right}(v)} \tag{14}$$

For an example of computation of $\overline{\mathrm{e}}$ and $\overline{\mathrm{Env}}$ see Figure 5.

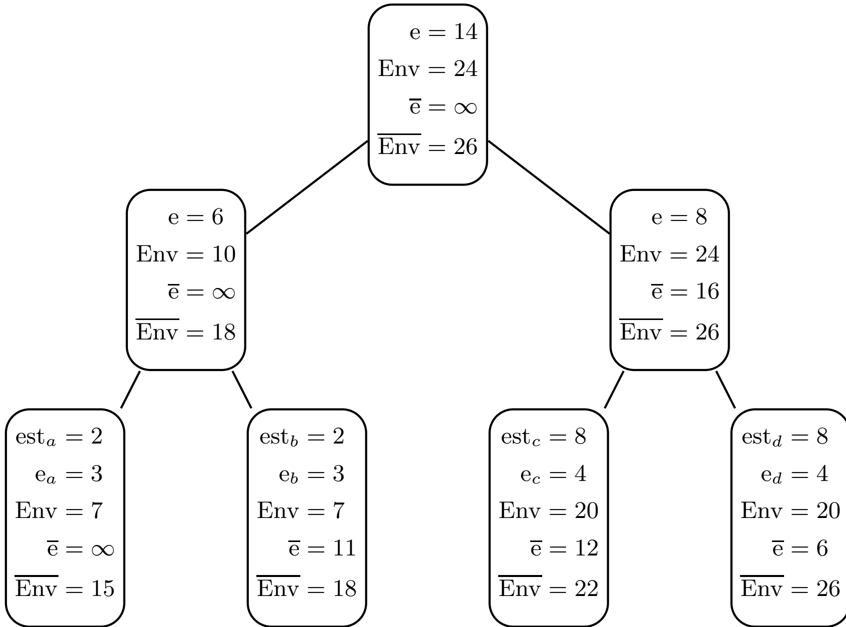

Fig. 5. Computation of $\overline{\mathrm{Env}}$ and $\overline{\mathrm{e}}$ for $\Theta = \{a, b, c, d\}$ from Figure 3. Notice that from the nodes representing activities c and d we can conclude that $\overline{\mathrm{e}}_c \leq 6$ and $\overline{\mathrm{e}}_d \leq 6$. In case of activity d because of $\overline{\mathrm{e}}$ in the node, in case of activity c because of $\overline{\mathrm{Env}}$ in the node as will be described by (15).

Formal proof of the recursive rules (11) – (14) will follow. But let us first explain for example construction of formula (13). If $\mathrm{e}_{\mathrm{right}(v)}$ is increased then it will cause also an increase of e_v by formula (6). However maximum feasible value

of e_v is $\overline{\mathrm{e}}_v$ and therefore the maximum feasible value of $\mathrm{e}_{\mathrm{right}(v)}$ has to fulfill the following formula:

$$\overline{\mathrm{e}}_{\mathrm{right}(v)} \leq \overline{\mathrm{e}}_v - \mathrm{e}_{\mathrm{left}(v)}$$

Similarly, increase of $\mathrm{e}_{\mathrm{right}(v)}$ can lead to the increase of Env_v by formula (7) but it cannot exceed the maximum feasible value $\overline{\mathrm{Env}}_v$. Therefore:

$$\overline{\mathrm{e}}_{\mathrm{right}(v)} \leq \overline{\mathrm{Env}}_v - \mathrm{Env}_{\mathrm{left}(v)}$$

Combining the these two formulas we get rule (13). The remaining three rules (11), (12) and (14) are constructed in a similar way.

Let us formally proof correctness of the rules (11) – (14). We start with the following lemma:

Lemma 1. *For a node w and its parent node v in a Θ-tree: if one of the values $\overline{\mathrm{Env}}_w$ and $\overline{\mathrm{e}}_w$ are not respected (that is $\mathrm{Env}_w > \overline{\mathrm{Env}}_w$ or $\mathrm{e}_w > \overline{\mathrm{e}}_w$) then at least one of the values $\overline{\mathrm{Env}}_v$, $\overline{\mathrm{e}}_v$ is not respected too.*

Proof. We will split the proof into two parts depending on whether w is left or right son of node v:

1. Case $w = \mathrm{left}(v)$. If $\overline{\mathrm{e}}_w$ is not respected then:

$$\begin{aligned} \mathrm{e}_w > \overline{\mathrm{e}}_w &\overset{(14)}{=} \overline{\mathrm{e}}_v - \mathrm{e}_{\mathrm{right}(v)} \\ \mathrm{e}_w + \mathrm{e}_{\mathrm{right}(v)} &> \overline{\mathrm{e}}_v \\ \mathrm{e}_v &> \overline{\mathrm{e}}_v \qquad \text{by (6)} \end{aligned}$$

   Therefore if $\overline{\mathrm{e}}_v$ is not respected then $\overline{\mathrm{e}}_v$ is not respected too.
   Similarly if $\overline{\mathrm{Env}}_w$ is not respected then:

$$\begin{aligned} \mathrm{Env}_w > \overline{\mathrm{Env}}_w &\overset{(12)}{=} \overline{\mathrm{Env}}_v - \mathrm{e}_{\mathrm{right}(v)} \\ \mathrm{Env}_w + \mathrm{e}_{\mathrm{right}(v)} &> \overline{\mathrm{Env}}_v \\ \max\{\mathrm{Env}_w + \mathrm{e}_{\mathrm{right}(v)},\ \mathrm{Env}_{\mathrm{right}(v)}\} &> \overline{\mathrm{Env}}_v \\ \mathrm{Env}_v &> \overline{\mathrm{Env}}_v \qquad \text{by (7)} \end{aligned}$$

   So if $\overline{\mathrm{Env}}_w$ is not respected then $\overline{\mathrm{Env}}_v$ is not respected too.
2. Case $w = \mathrm{right}(v)$. If $\overline{\mathrm{e}}_w$ is not respected then:

$$\mathrm{e}_w > \overline{\mathrm{e}}_w \overset{(13)}{=} \min\{\overline{\mathrm{Env}}_v - \mathrm{Env}_{\mathrm{left}(v)},\ \overline{\mathrm{e}}_v - \mathrm{e}_{\mathrm{left}(v)}\}$$

   Therefore
   (a) Either:

$$\begin{aligned} \mathrm{e}_w &> \overline{\mathrm{Env}}_v - \mathrm{Env}_{\mathrm{left}(v)} \\ \mathrm{Env}_{\mathrm{left}(v)} + \mathrm{e}_w &> \overline{\mathrm{Env}}_v \\ \mathrm{Env}_v &> \overline{\mathrm{Env}}_v \qquad \text{by (7)} \end{aligned}$$

   And so $\overline{\mathrm{Env}}_v$ is not respected.

(b) Or:

$$\begin{aligned} \mathrm{e}_w &> \overline{\mathrm{e}}_v - \mathrm{e}_{\mathrm{left}(v)} \\ \mathrm{e}_{\mathrm{left}(v)} + \mathrm{e}_w &> \overline{\mathrm{e}}_v \\ \mathrm{e}_v &> \overline{\mathrm{e}}_v \qquad \text{by (6)} \end{aligned}$$

And so $\overline{\mathrm{e}}_v$ is not respected.

Finally if $\overline{\mathrm{Env}}_w$ is not respected then:

$$\mathrm{Env}_w > \overline{\mathrm{Env}}_w \overset{(11)}{=} \overline{\mathrm{Env}}_v$$

Therefore

$$\mathrm{Env}_v \overset{(7)}{=} \max\{\mathrm{Env}_{\mathrm{left}(v)} + \mathrm{e}_w,\ \mathrm{Env}_w\} \geq \mathrm{Env}_w > \overline{\mathrm{Env}}_v$$

And thus $\overline{\mathrm{Env}}_v$ is not respected. □

A consequence of this lemma is:

Proposition 3. *Let $i \in \mathrm{LCut}(T, j)$ and let v be a leaf node representing activity i in Θ-tree for $\mathrm{LCut}(T, j)$. If $\mathrm{e}_i > \min\left\{\overline{\mathrm{e}}_v,\ \overline{\mathrm{Env}}_v - C\,\mathrm{est}_i\right\}$ then there is an overload and therefore the problem is unfeasible.*

Proof. If $\mathrm{e}_i > \min\left\{\overline{\mathrm{e}}_v,\ \overline{\mathrm{Env}}_v - C\,\mathrm{est}_i\right\}$ then it means that either $\overline{\mathrm{e}}_v$ or $\overline{\mathrm{Env}}_v$ in the node v is not respected. By the previous lemma it means that at least one of these values is not respected also in parent node of v. And so we continue this way to the root node r and prove that $\overline{\mathrm{e}}_r$ or $\overline{\mathrm{Env}}_r$ is not respected.

However for root node r, $\overline{\mathrm{e}}_r = \infty$ by (10) therefore $\overline{\mathrm{e}}_r$ has to be respected. The conclusion is that $\overline{\mathrm{Env}}_r$ is not respected and therefore:

$$\begin{aligned} \mathrm{Env}_r > \overline{\mathrm{Env}}_r &\overset{(9)}{=} C\,\mathrm{lct}_j \\ \mathrm{Env}\left(\mathrm{LCut}\left(T, j\right)\right) &> C\,\mathrm{lct}_j \end{aligned}$$

So there is overload by Proposition 1. □

The proposition above gives as an upper bound for maximum energy available for each activity $i \in \mathrm{LCut}(T, j)$:

$$\overline{\mathrm{e}}_i \leq \min\left\{\overline{\mathrm{e}}_v,\ \overline{\mathrm{Env}}_v - C\,\mathrm{est}_i\right\} \tag{15}$$

Notice that for example on Figures 3 and 5 the formula (15) gives $\overline{\mathrm{e}}_c = 6$ and $\overline{\mathrm{e}}_d = 6$ and therefore by (2) $\overline{\mathrm{p}}_c = 3$ and $\overline{\mathrm{p}}_d = 3$.

The basic idea of the algorithm follows: we iterate over all activities $j \in T$ and for each j we build Θ-tree representing $\mathrm{LCut}(T, j)$ by adding new nodes into the Θ-tree from the previous iteration. In each Θ-tree we propagate the maximum energy envelope $C\,\mathrm{lct}_j$ from the root to leave nodes and assign maximum energies to activities $i \in \mathrm{LCut}(T, j)$ according to formula (15). First version of the algorithm with time complexity $\mathcal{O}(n^2)$ is provided by Algorithm 1.2. Note that this is not the $\mathcal{O}(n^2)$ algorithm by Wim Nuijten, for better understanding we start with $\mathcal{O}(n^2)$ algorithm and then speed it up to $\mathcal{O}(n \log n)$.

The algorithm uses two procedures:

– `push_down(`v`)` pushes the values $\overline{\text{Env}}_v$ and $\overline{\text{e}}_v$ from the node v down the tree using the rules (11) – (14).
– `set_energy_max(`i`)` sets maximum energy $\overline{\text{e}}_i$ of the activity i using formula (15).

Algorithm 1.2. Maximum energy propagation in $\mathcal{O}(n^2)$

```
 1  Θ := ∅;
 2  for j ∈ T in non-decreasing order of lct_j do begin
 3     Θ := Θ ∪ {j};
 4     if Env(Θ) > C lct_j then
 5        fail; {No solution exists}
 6     ‾Env‾_Θ := C lct_j;
 7     for nodes v in Θ-tree in non-decreasing order of their depth do
 8        push_down(v);
 9     for i ∈ Θ do
10        set_energy_max(i);
11  end;
```

Time complexity of this algorithm is $\mathcal{O}(n^2)$ because the inner cycles on lines 7 – 8 and 9 – 10 have time complexity $\mathcal{O}(n)$. In the following we will show how to improve the time complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$.

The key observation is that it is not necessary to push values $\overline{\text{e}}_v$ and $\overline{\text{Env}}_v$ down to leaves immediately. The values $\overline{\text{e}}_v$ and $\overline{\text{Env}}_v$ stays valid until new node is inserted into the subtree of v. Therefore it is possible to postpone `push_down(`v`)` until new node is inserted somewhere under the node v.

Current procedure `push_down(`v`)` simply overwrites values $\overline{\text{e}}$ and $\overline{\text{Env}}$ in children nodes of v. However that is no longer possible in the new algorithm because children nodes may contain information which was not pushed down yet. Therefore it is necessary create new procedure `push_down2(`v`)` which implements modified rules (11) – (14):

$$\overline{\text{Env}}_{\text{right}(v)} := \min\left\{\overline{\text{Env}}_v,\ \overline{\text{Env}}_{\text{right}(v)}\right\} \tag{16}$$

$$\overline{\text{Env}}_{\text{left}(v)} := \min\left\{\overline{\text{Env}}_v - \text{e}_{\text{right}(v)},\ \overline{\text{Env}}_{\text{left}(v)}\right\} \tag{17}$$

$$\overline{\text{e}}_{\text{right}(v)} := \min\left\{\overline{\text{Env}}_v - \text{Env}_{\text{left}(v)},\ \overline{\text{e}}_v - \text{e}_{\text{left}(v)},\ \overline{\text{e}}_{\text{right}(v)}\right\} \tag{18}$$

$$\overline{\text{e}}_{\text{left}(v)} := \min\left\{\overline{\text{e}}_v - \text{e}_{\text{right}(v)},\ \overline{\text{e}}_{\text{left}(v)}\right\} \tag{19}$$

The values $\overline{\text{e}}_v$ and $\overline{\text{Env}}_v$ must be initialized to:

$$\overline{\text{e}}_v := \infty$$

$$\overline{\text{Env}}_v := \infty$$

The postponed calls of push_down2(v) must be executed just before new node is added into the tree. The most suitable place is to replace $\Theta := \Theta \cup \{j\}$ on line 3 by procedure add(Θ, j) which will make necessary postponed calls.

For simplicity, lets assume that the shape of the tree is fixed and no re-balancing occurs during the addition of new node into the tree[3]. The procedure add(Θ, j) has following steps:

1. Find a node w under which new node will be inserted.
2. Call push_down2(v) on all nodes v on the path from the root to w, and then reset these nodes to:

$$\overline{\mathrm{e}}_v := \infty$$
$$\overline{\mathrm{Env}}_v := \infty$$

3. Add new node in the tree, fill the leaf representing j by data about the activity j.
4. Recompute values e_v and Env_v on the path from this leaf to the root.

The resulting Algorithm 1.3 has worst case time complexity $\mathcal{O}(n \log n)$.

Algorithm 1.3. Maximum energy propagation in $\mathcal{O}(n \log n)$

```
1   Θ := ∅;
2   for j ∈ T in non-decreasing order of lct_j do begin
3     add(Θ, j);
4     if Env(Θ) > C lct_j then
5       fail; {No solution exists}
6     ‾Env‾_Θ := C lct_j;
7   end;
8   for nodes v in Θ-tree in non-decreasing order of their depth do
9     push_down2(v);
10  for i ∈ T do
11    set_energy_max(i);
```

5 Optional Activities

As mentioned in the introduction, IBM ILOG CP Optimizer 2.0 [1] introduce a new variable type designed for scheduling – interval variable. The difference between activity (as we used this word in this paper) and an interval variable is that an interval variable by itself does not require any resource. An activity is created when an interval variable is constrained to require a resource. One interval variable may require more than one resource, in this case the interval variable is associated with several activities and all of them share their start times and end times.

[3] This can be achieved for example by computing perfectly balanced tree of all activities in advance.

Interval variable in IBM ILOG CP Optimizer has very important aspect: it can be declared as optional, that is, it may or may not be present in the solution. In this case all its activities are also optional.

Typical use of optional interval variables are alternative between two different actions. This can be easily modeled by two optional interval variables and a constraint that exactly one of them is present in the solution. Both optional intervals can require some discrete cumulative resource during their execution, however the interval which is not present in the solution does not affect any resource. For more details about optional interval variables see [4, 1].

From the point of view of the resource, there are some optional activities which may or may not be executed on the resource, this is yet to be decided. During the search an optional activity may become:

A) preset if:
   1) we decide to execute it as a search decision,
   2) or if we proved (by propagation) that no other alternative is possible,
B) absent if:
   1) we decided to not to execute it by a search decision,
   2) or if we proved (by propagation) that execution of this activity is not possible.

The algorithm Max Energy presented in this paper can do the propagation B2) above. This is very important propagation because usually any propagation on optionality status has a big impact on other variables.

How to use Algorithm 1.3 with optional activities? First observe that from the point of view of discrete cumulative resource, there is no difference between absent activity and activity with zero energy (that is, activity with zero duration or zero capacity requirement). Zero-energy activity can be processed anytime even though the resource is already full.

The idea is to use Algorithm 1.3 directly without any modification, but on modified input data. If an activity is optional then (just for the algorithm) its minimum energy is set to zero. This way, optional activity cannot influence any other activity, however upper bound $\overline{\mathrm{e}}_i$ for energy computed by the algorithm is valid also for optional activity i. Furthermore if $\overline{\mathrm{e}}_i < \mathrm{e}_i$ we can deduce that the activity i cannot be present in the solution.

6 Conclusions and Further Work

This paper presents a new Max Energy propagation algorithm which updates maximum durations and maximum capacity requirements on discrete cumulative resource with optional activities. The algorithm has better time complexity ($\mathcal{O}(n \log n)$ versus $\mathcal{O}(n^2)$) than old never published algorithm by Wim Nuijten. Experiments show that the new algorithm begin to be faster than the old one for n around 10. The algorithm is used by IBM ILOG CP Optimizer [1] since version 2.0.

References

[1] IBM ILOG CP Optimizer, http://www.ilog.com/products/cpoptimizer/
[2] Mercier, L., van Hentenryck, P.: Edge finding for cumulative scheduling. Informs Journal of Computing 20(1), 143–153 (2008)
[3] Philippe Baptiste, C.L.P., Nuijten, W.: Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems. Kluwer Academic Publishers, Dordrecht (2001)
[4] Philippe Laborie, J.R.: Reasoning with conditional time-intervals. In: Wilson, D., Lane, H.C. (eds.) Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference, Coconut Grove, Florida, USA, May 15-17, 2008, pp. 555–560. AAAI Press, Menlo Park (2008)
[5] Schutt, A., Wolf, A., Schrader, G.: Not-first and not-last detection for cumulative scheduling in $O(n^3 \log n)$. In: Umeda, M., Wolf, A., Bartenstein, O., Geske, U., Seipel, D., Takata, O. (eds.) INAP 2005. LNCS, vol. 4369, pp. 66–80. Springer, Heidelberg (2006)
[6] 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, KTIML MFF, Universita Karlova, Malostranské náměstí 2/25, 118 00 Praha 1, Czech Republic (2007), http://vilim.eu/petr/disertace.pdf
[7] Wolf, A., Schrader, G.: $O(n \log n)$ overload checking for the cumulative constraint and its application. In: Umeda, M., Wolf, A., Bartenstein, O., Geske, U., Seipel, D., Takata, O. (eds.) INAP 2005. LNCS, vol. 4369, pp. 88–101. Springer, Heidelberg (2006)