Timetable Edge Finding Filtering Algorithm for Discrete Cumulative Resources

Petr Vilím

IBM, V Parku 2294/4
148 00 Praha 4 - Chodov, Czech Republic
petr_vilim@cz.ibm.com

Abstract. Edge Finding filtering algorithm is one of the reasons why Constraint Programming is a successful approach in the scheduling domain. However edge finding for cumulative resources was never as successful as edge finding for disjunctive resources. This paper presents a new variant of the edge finding algorithm which improves filtering by taking into account minimum capacity profile - a data structure known from timetabling algorithm. In comparison with standard and extended edge finding algorithms the new algorithm is stronger but it may need more iterations in order to reach the fixpoint. Time complexity of the algorithm is $\mathcal{O}(n^2)$ where n is number of activities on the resource. We also propose further improvement of the filtering by incorporating some ideas from not-first/not-last and energetic reasoning algorithms. The filtering power of the algorithm is tested on computation of destructive lower bounds for 438 open RCPSP problems. For 169 of them we improve current best lower bound, in 9 cases backtrack free.

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

1 Introduction

This paper focuses on discrete cumulative resource – an abstraction of manpower, electricity, machinery or any other (renewable) resource which is used to perform activities (tasks to be scheduled). Although the resource can be used by several activities simultaneously, total resource capacity used at any time cannot exceed capacity limit C. In a constraint programming framework we usually associate a constraint with each resource. The task of this resource constraint is to remove inconsistent values from temporal variables associated with activities. For the rest of the paper we will concentrate on propagation for a single resource.

A discrete cumulative resource is characterized by maximum capacity of the resource $C \in \mathbb{N}$ and a set T of n activities, $n = |T|$. Each activity has the following attributes:

- the earliest start time $\mathrm{est}_i \in \mathbb{N}$,
- the latest completion time (deadline) $\mathrm{lct}_i \in \mathbb{N}$,
- the processing time (duration) $\mathrm{p}_i \in \mathbb{N}$ (a constant),

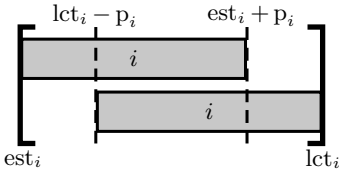


Fig. 1. Two extreme positions of activity i. Regardless of its position, activity i always uses the resource during $[\mathrm{lct}_i - \mathrm{p}_i, \mathrm{est}_i + \mathrm{p}_i]$.

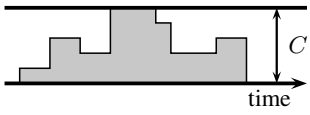


Fig. 2. Accumulated minimum capacity profile.

- the required capacity $\mathrm{c}_i \in \mathbb{N}$ (a constant),
- and energy of the activity $\mathrm{e}_i = \mathrm{c}_i\,\mathrm{p}_i$ (a constant).

Activities are assumed to be non-preemptive: once the processing of activity i starts at time t then it must continue without preemption until $t + \mathrm{p}_i$. For each activity we maintain a decision variable for the start time of the activity with domain $[\mathrm{est}_i, \mathrm{lct}_i - \mathrm{p}_i]$. The aim of the resource constraint is to remove inconsistent values from this domain by increasing est_i and decreasing lct_i.

1.1 Related Works

This section reviews some of the existing techniques to propagate cumulative resource constraint.

Timetabling The idea of timetabling is to look for activities i such that $\mathrm{lct}_i - \mathrm{p}_i < \mathrm{est}_i + \mathrm{p}_i$, see Figure 1. Such activities necessarily use the resource during interval $[\mathrm{lct}_i - \mathrm{p}_i,\ \mathrm{est}_i + \mathrm{p}_i]$. By aggregating these intervals we compute a minimum capacity profile (a timetable) which shows minimum resource usage over time (Figure 2). Typically, the minimum capacity profile is maintained during the search and used to detect infeasibility and also to update time bounds of activities. For more information see [6, chapters 3.3.1 and 2.1.1].

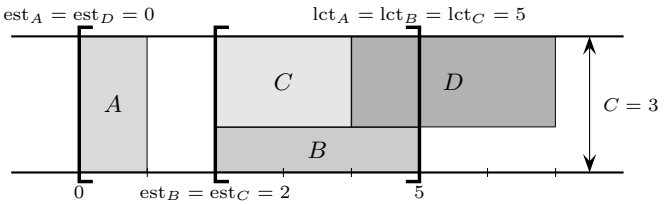
Edge Finding and Extended Edge Finding Unlike timetabling, edge finding propagation is based on reasoning about a set of activities. Let us consider a set of activities $\Omega \subseteq T$. For Ω we define earliest starting time est_Ω, latest completion time lct_Ω and energy e_Ω as:

$$\mathrm{est}_\Omega = \min\{\mathrm{est}_i,\ i \in \Omega\} \qquad \mathrm{e}_\Omega = \sum_{i \in \Omega} \mathrm{e}_i$$
$$\mathrm{lct}_\Omega = \max\{\mathrm{lct}_i,\ i \in \Omega\}$$

The total energy available for Ω is $C\,(\mathrm{lct}_\Omega - \mathrm{est}_\Omega)$. If Ω requires more energy then there is no solution (this is called overload rule):

$$\forall \Omega \subseteq T : (\ \ C\,(\mathrm{lct}_\Omega - \mathrm{est}_\Omega) < \mathrm{e}_\Omega \ \ \Rightarrow \ \ \text{fail}\ \ )$$

Edge finding is also able to update temporal bounds of activities. Informally speaking, it checks whether scheduling an activity i at its earliest start time est_i would lead to overload as described above. If it is the case then est_i is



act.	est	lct	p	c
A	0	5	1	3
B	2	5	3	1
C	2	5	2	2
D	0	99	4	2

Fig. 3. An example from [11]: resource with capacity $C = 3$ and activities $\{A, B, C, D\}$. The table on the right summarizes attributes of these activities. Edge finding updates est_D from 0 to 4 because scheduling D at 0 would lead to overflow in interval $[0, 5]$.

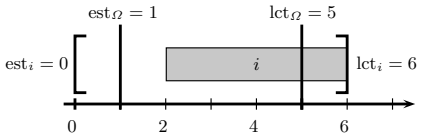


Fig. 4. Activity i with $\text{est}_i = 0$, $\text{lct}_i = 6$, $\text{p}_i = 4$ and $\text{c}_i = 1$. Activity i requires at least 3 energy units during $[1, 5]$. However edge finding does not take them into account.

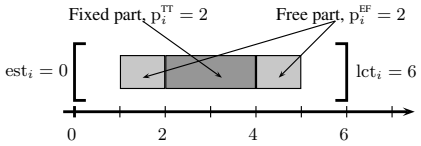


Fig. 5. Fixed and free parts of activity i with $\text{est}_i = 0$, $\text{lct}_i = 6$, $\text{p}_i = 4$.

updated. For an example see Figure 3, details are described in Section 4. There is also a symmetrical algorithm to update lct_i. Paper [5] provides algorithms for both standard and extended edge finding algorithms with time complexity $\mathcal{O}(kn^2)$ (k is number of distinct capacity demands $k = |\{c_i,\ i \in T\}|$). There is also a standard (not extended) edge finding algorithm with time complexity $\mathcal{O}(kn \log n)$ in [11]. There are also independent attempts to design edge finding algorithms with better time complexities by Roger Kameugne.

Energetic Reasoning Edge finding is not perfectly accurate in computation of energy requirement during the interval $[\text{est}_\Omega, \text{lct}_\Omega]$. In particular, e_Ω does not take into account activities that only partially overlap with $[\text{est}_\Omega, \text{lct}_\Omega]$, see Figure 4. Energetic reasoning (also called CNP-ER) is able to take such partial overlaps into account. It also considers more intervals than only $[\text{est}_\Omega, \text{lct}_\Omega]$. Chapter 3.3.6 in [6] presents an algorithms to detect infeasibility and to update temporal bounds, their time complexities are $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$.

Dominance Relations Chapter 4.2.4 in [6] contains a detailed analysis of dominance relations between different filtering algorithms for cumulative resources. The conclusion is that energetic reasoning is stronger than both timetabling and edge finding. However edge finding does not dominate timetabling and vice versa. Therefore edge finding can be improved by taking into account the timetable as we suggest in this paper. Energetic reasoning is still stronger than the proposed algorithm but slower (time complexity $\mathcal{O}(n^3)$ versus $\mathcal{O}(n^2)$). What we propose is a good trade off between filtering power and speed.

Note that for practical reasons the proposed algorithm is not designed to subsume the timetable algorithm: it is faster to do some propagation by the timetable algorithm.

2 TimeTable Edge Finding

The idea is to improve computation of the energy consumed during $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$ using information from the timetable. Let us consider again the example on Figure 4. From timetabling point of view, activity i contributes to the minimum capacity profile by 2 energy units on interval $[2, 4]$. By using timetable in the edge finding we will be able to take the 2 energy units into account and therefore improve filtering. Note that it is still less than the 3 energy units detected by energetic reasoning but often the values are the same.

In order to use timetable we split each activity into two parts: *fixed* part (counted in the timetable) and remaining *free* part. See Figures 1 and 5. The split is done in the following way. For each activity i we compute its fixed duration $\mathrm{p}_i^{\mathrm{TT}}$ and free duration $\mathrm{p}_i^{\mathrm{EF}}$ as:

$$\mathrm{p}_i^{\mathrm{TT}} = \max\left(0,\ \mathrm{est}_i + \mathrm{p}_i - (\mathrm{lct}_i - \mathrm{p}_i)\right) \qquad \mathrm{p}_i^{\mathrm{EF}} = \mathrm{p}_i - \mathrm{p}_i^{\mathrm{TT}}$$

Fixed and free parts are handled differently:

Fixed Part If $\mathrm{p}^{\mathrm{TT}} = 0$ then the fixed part is empty and we ignore it. Otherwise we increase the minimum consumption level in the timetable on interval $[\mathrm{lct}_i - \mathrm{p}_i, \mathrm{est}_i + \mathrm{p}_i]$ by c_i.

Free Part Empty free parts ($\mathrm{p}_i^{\mathrm{EF}} = 0$) are ignored. Activities with non-empty free parts form a set $T^{\mathrm{EF}} = \{i,\ i \in T\ \&\ \mathrm{p}^{\mathrm{EF}} > 0\}$. We also define the energy of the free part as $\mathrm{e}_i^{\mathrm{EF}} = \mathrm{c}_i\,\mathrm{p}_i^{\mathrm{EF}}$. We can consider a free part of activity i as a separate activity with the same earliest start time est_i and latest completion time lct_i as the original activity i. However, unlike the original activity, the free part is preemptive: it can (and must) be suspended during $[\mathrm{lct}_i - \mathrm{p}_i, \mathrm{est}_i + \mathrm{p}_i]$.

2.1 Timetable

Timetable records the minimum capacity consumption at each time point t. In other words this data structure represents a function $\mathrm{TT}(t)$ such that $\mathrm{TT}(t)$ is the sum of capacities of all fixed parts which overlap time t (Figure 2).

For the presented algorithm we need to know how much energy is stored in the timetable for interval $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$. To compute it we introduce functions $\mathrm{ttAfterEst}(i)$ and $\mathrm{ttAfterLct}(i)$, they compute the total energy stored in the timetable after est_i and after lct_i:

$$\mathrm{ttAfterEst}(i) = \sum_{t \in \mathbb{N} \wedge t \geq \mathrm{est}_i} \mathrm{TT}(t) \qquad \mathrm{ttAfterLct}(i) = \sum_{t \in \mathbb{N} \wedge t \geq \mathrm{lct}_i} \mathrm{TT}(t)$$

Let us consider set $\Omega \subseteq T$ and activities $a, b \in \Omega$ such that $\mathrm{est}_a = \mathrm{est}_\Omega$ and $\mathrm{lct}_b = \mathrm{lct}_\Omega$. Then the energy stored in the timetable for interval $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$ is $\mathrm{ttAfterEst}(a) - \mathrm{ttAfterLct}(b)$. For simplicity we define also:

$$\mathrm{ttAfterEst}(\Omega) = \mathrm{ttAfterEst}(a) \ \text{ where } a \in \Omega \text{ and } \mathrm{est}_a = \mathrm{est}_\Omega$$
$$\mathrm{ttAfterLct}(\Omega) = \mathrm{ttAfterLct}(b) \ \text{ where } b \in \Omega \text{ and } \mathrm{lct}_b = \mathrm{lct}_\Omega$$

Algorithm 1. Overload Checking

```
1  for b ∈ T^EF do begin
2    eEF := 0;
3    for a ∈ T^EF in non-increasing order by est_a do
4      if lct_a ≤ lct_b then begin
5        eEF := eEF + e_a^EF;
6        if C(lct_b − est_a) < eEF + ttAfterEst[a] − ttAfterLct[b] then
7          fail;
8      end;
9  end;
```

Functions ttAfterEst(i) and ttAfterLct(i) can be computed from function TT(t) in $\mathcal{O}(n \log n)$ as follows. We sort activities into four lists: according to $\mathrm{est}_i + \mathrm{p}_i$, $\mathrm{lct}_i - \mathrm{p}_i$, est_i and lct_i. Then we sweep over all these events in anti-chronological order. During the sweep we maintain total energy stored in the timetable after the current event. The result of the computation is stored in arrays `ttAfterEst` and `ttAfterLct`.

2.2 Overload Checking

We start by the algorithm for checking infeasibility. Let's consider set $\Omega \subseteq T^{\mathrm{EF}}$. Energy of free parts of activities in Ω is $\mathrm{e}^{\mathrm{EF}}_{\Omega} = \sum_{i \in \Omega} \mathrm{e}^{\mathrm{EF}}_i$. Therefore minimum energy consumption by both fixed and free parts during $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$ is $\mathrm{e}^{\mathrm{EF}}_{\Omega} + \mathrm{ttAfterEst}\,(\Omega) - \mathrm{ttAfterLct}\,(\Omega)$. However energy available in $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$ is $C\,(\mathrm{lct}_\Omega - \mathrm{est}_\Omega)$. Therefore remaining energy "reserve" in $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$ is:

$$\mathrm{reserve}(\Omega) = C\,(\mathrm{lct}_\Omega - \mathrm{est}_\Omega) - (\mathrm{e}^{\mathrm{EF}}_{\Omega} + \mathrm{ttAfterEst}\,(\Omega) - \mathrm{ttAfterLct}\,(\Omega))$$

If the reserve is negative then the problem is infeasible:

$$\forall \Omega \subseteq T^{\mathrm{EF}} : \ (\ \mathrm{reserve}(\Omega) < 0 \ \ \Rightarrow \mathrm{fail})$$

Clearly it is not necessary to check all sets $\Omega \subseteq T^{\mathrm{EF}}$. Let us consider two sets Ω_1 and Ω_2 such that:

$$\mathrm{est}_{\Omega_1} = \mathrm{est}_{\Omega_2} \qquad \mathrm{lct}_{\Omega_1} = \mathrm{lct}_{\Omega_2} \qquad \mathrm{e}^{\mathrm{EF}}_{\Omega_1} > \mathrm{e}^{\mathrm{EF}}_{\Omega_2}$$

Then it is enough to check only set Ω_1. Therefore the rule must be checked only for sets Ω with the following property:

$$\forall i \in T^{\mathrm{EF}} : \mathrm{est}_i \geq \mathrm{est}_\Omega \ \ \& \ \ \mathrm{lct}_i \leq \mathrm{lct}_\Omega \ \ \Rightarrow \ \ i \in \Omega$$

Such sets Ω are traditionally called task intervals.

Algorithm 1 checks infeasibility by the rule above. The idea is to pick a task $b \in T^{\mathrm{EF}}$ and iterate over sets Ω such that $\mathrm{lct}_\Omega = \mathrm{lct}_b$. Time complexity of the algorithm is $\mathcal{O}(n^2)$.

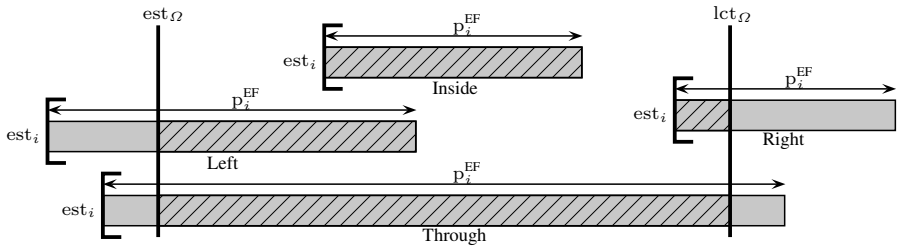


Fig. 6. Different relative positions of activity $i \in T^{\mathrm{EF}}$ and set Ω such that scheduling i on est_i increases energy consumption in $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$. The increase is marked by hatched lines.

Position	Characterization	Additional consumption
Inside	$\mathrm{est}_\Omega \leq \mathrm{est}_i \ \&\ \mathrm{est}_i + \mathrm{p}_i^{\mathrm{EF}} \leq \mathrm{lct}_\Omega$	$\mathrm{e}_i^{\mathrm{EF}}$
Right	$\mathrm{est}_\Omega < \mathrm{est}_i < \mathrm{lct}_\Omega < \mathrm{est}_i + \mathrm{p}_i^{\mathrm{EF}}$	$\mathrm{c}_i(\mathrm{lct}_\Omega - \mathrm{est}_i)$
Left	$\mathrm{est}_i < \mathrm{est}_\Omega < \mathrm{est}_i + \mathrm{p}_i^{\mathrm{EF}} < \mathrm{lct}_\Omega$	$\mathrm{c}_i(\mathrm{est}_i + \mathrm{p}_i^{\mathrm{EF}} - \mathrm{est}_\Omega)$
Through	$\mathrm{est}_i \leq \mathrm{est}_\Omega \ \&\ \mathrm{lct}_\Omega \leq \mathrm{est}_i + \mathrm{p}_i^{\mathrm{EF}}$	$\mathrm{c}_i(\mathrm{lct}_\Omega - \mathrm{est}_\Omega)$
Out	Otherwise	0

Table 1. Additional consumption of activity $i \notin \Omega$ in $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$ when scheduled on est_i.

2.3 Time Bound Adjustment Rule

Value est_i is invalid if scheduling activity i on est_i would cause overload as described above. In this case est_i can be updated. To check such potential overloads, we need to compute how much additional energy activity $i \in T^{\mathrm{EF}}$ would require during $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$ if i is scheduled on est_i. If $i \in \Omega$ then the whole energy of i is already counted so we concentrate only on the case $i \notin \Omega$. For $i \notin \Omega$ we distinguish four relative positions of Ω and free part of i such that scheduling the free part of i on est_i increases energy consumption in $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$. See Figure 6 and Table 1.

Let function $\mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i)$ denote additional energy consumption by activity i in $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$ as defined by Table 1. Scheduling activity i on est_i causes overload with set Ω if:

$$\mathrm{reserve}(\Omega) < \mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i)$$

In this case current est_i can be updated. In the following we will show how to compute this update.

We start by computation of the maximum duration that activity i can spend inside $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$. This duration has two parts:

1. **Mandatory part:** the part of i which is in the timetable and which overlaps with $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$. It cannot move and therefore it must stay inside $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$.
2. **Optional part:** this is the maximum of free duration $\mathrm{p}_i^{\mathrm{EF}}$ which can still fit inside $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$ considering $\mathrm{reserve}(\Omega)$.

The mandatory part is intersection of intervals $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$ and $[\mathrm{lct}_i - \mathrm{p}_i, \mathrm{est}_i + \mathrm{p}_i]$. Its length is:

$$\begin{aligned}\mathrm{mandatoryIn}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i) = \\ = \max\left(0, \min\left(\mathrm{lct}_\Omega, \mathrm{est}_i + \mathrm{p}_i\right) - \max\left(\mathrm{est}_\Omega, \mathrm{lct}_i - \mathrm{p}_i\right)\right)\end{aligned}$$

Maximum length of the optional part is:

$$\mathrm{maxAddIn}(\Omega, i) = \left\lfloor \frac{\mathrm{reserve}(\Omega)}{c_i} \right\rfloor$$

The remaining duration of i must be spent after lct_Ω. Therefore est_i can be updated to the following new value:

$$\mathrm{est}_i := \mathrm{lct}_\Omega - \mathrm{mandatoryIn}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i) - \mathrm{maxAddIn}(\Omega, i)$$

The full propagation rule is:

$$\begin{aligned}\forall \Omega \subset T^{\mathrm{EF}},\ \forall i \in T^{\mathrm{EF}} \setminus \Omega : \mathrm{reserve}(\Omega) < \mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i) \quad \Rightarrow \\ \mathrm{est}_i := \mathrm{lct}_\Omega - \mathrm{mandatoryIn}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i) - \mathrm{maxAddIn}(\Omega, i)\end{aligned}$$

There is quite a big difference between the rule above and standard edge finding (or extended edge finding) propagation rule. Standard edge finding tries to find best subset $\Omega' \subseteq \Omega$ to immediately achieve the best update of est_i. The rule above fixes only potential overflow with Ω. Unlike standard edge finding it doesn't notice that the new est_i can be still invalid with respect to some $\Omega' \subset \Omega$. The proposed rule simply assumes that the algorithm will run again and if there is still a problem with the new est_i then it will be updated again. In this respect it may take more iterations for this algorithm to reach the fixpoint. In practice it is not a big problem as discussed later in section 2.5.

2.4 Time Bound Adjustment Algorithm

First of all, notice that to achieve maximum propagation it is enough to concentrate on sets Ω in the form of task intervals. The reason is the same as for the overload rule.

The idea of the algorithm is to iterate over sets Ω in the form of task intervals in the same way as the overload algorithm does. For each set Ω we also compute maximum value of $\mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i)$ over all $i \in T^{\mathrm{EF}} \setminus \Omega$. Let ι be activity i such that $\mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i)$ is maximal. We distinguish two cases:

1. If $\mathrm{reserve}(\Omega) \geq \mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, \iota)$ then activity ι cannot be updated by Ω. And because $\mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, \iota)$ is maximal, Ω cannot update any other activity $i \in T^{\mathrm{EF}} \setminus \Omega$ neither. Thus we can continue with the next set Ω.
2. If $\mathrm{reserve}(\Omega) < \mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, \iota)$ then the algorithm updates est_ι. There could be more activities than only ι which could be updated by Ω. However the algorithm does not update them, they will be updated in the next iteration of the algorithm. This is the second reason why the algorithm needs more iterations to reach the fixpoint. This issue is discussed in Section 2.5.

It remains to show how to maintain ι during the algorithm. The value of function $\mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i)$ depends on the relative position of i and Ω as described in Table 1. To maintain the maximum of $\mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i)$ the algorithm is split into three phases, each phase deals only with some relative positions. In the following we describe each phase separately. For simplicity we will assume in the description that there are no duplicates in est_i and lct_i. However the algorithm is sound even in case of duplicates.

Inside and Right For Inside and Right positions we iterate over activities a and b the same way as we do in the Algorithm 1 for overload checking (lines 3–7 in Algorithm 2). That is, in outer loop we iterate over activity b (in arbitrary order) and in the inner loop we iterate over activity a in non-increasing order by est_a. Activities a such that $\mathrm{est}_a \geq \mathrm{lct}_b$ are skipped[1]. If $\mathrm{lct}_a \leq \mathrm{lct}_b$ then activities a and b define set $\Omega = \{j,\ j \in T^{\mathrm{EF}}\ \&\ \mathrm{est}_a \leq \mathrm{est}_j\ \&\ \mathrm{lct}_j \leq \mathrm{lct}_b\}$, its energy $\mathrm{e}^{\mathrm{EF}}_\Omega$ is stored in variable eEF.

The set of activities in Inside or Right position with Ω is $I = \{i,\ i \in T^{\mathrm{EF}}\ \&\ \mathrm{est}_\Omega \leq \mathrm{est}_i < \mathrm{lct}_\Omega < \mathrm{lct}_i\}$. Therefore as we iterate over activities a, each activity a is either put into Ω (if $\mathrm{lct}_a \leq \mathrm{lct}_b$, line 9) or put into I (if $\mathrm{lct}_a > \mathrm{lct}_b$). Each time when i is added into I we have to recompute ι. According to Table 1 value $\mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i)$ for positions Inside and Right can be computed as $\min\left(\mathrm{e}^{\mathrm{EF}}_i, \mathrm{c}_i\left(\mathrm{lct}_b - \mathrm{est}_i\right)\right)$. Notice that this value does not depend at all on activity a. This justifies computation of ι on lines 10–11.

Finally, if $\mathrm{reserve}(\Omega)$ is less then $\mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, \iota)$ then est_ι is updated (lines 13 and 14).

Through Again there are two nested cycles over activities a and b which define set $\Omega = \{j,\ j \in T^{\mathrm{EF}}\ \&\ \mathrm{est}_a \leq \mathrm{est}_j\ \&\ \mathrm{lct}_j \leq \mathrm{lct}_b\}$. Energy $\mathrm{e}^{\mathrm{EF}}_\Omega$ is stored in variable eEF. However in comparison with the previous phase we iterate over activities a in reverse order. That is, we gradually remove activities a from Ω.

As we iterate over activities a we first check whether $a \in \Omega$ (line 21). If $a \in \Omega$ then we remove it from Ω (line 25) but just before the removal we check whether ι can be updated by Ω (lines 22–24). Furthermore if $\mathrm{est}_a + \mathrm{p}^{\mathrm{EF}}_a \geq \mathrm{lct}_b$ (line 27) then activity a is in Through position with all following sets Ω. In this case ι needs to be recomputed. According to Table 1 value $\mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i)$ for case Through is $\mathrm{c}_i(\mathrm{lct}_\Omega - \mathrm{est}_\Omega)$. Therefore the maximum value of $\mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i)$ over all activities in Through position with Ω is achieved by the activity with the maximum capacity (lines 27–28).

Left One more time, activities a and b define set Ω such that $\mathrm{est}_\Omega = \mathrm{est}_a$ and $\mathrm{lct}_\Omega = \mathrm{lct}_b$; energy $\mathrm{e}^{\mathrm{EF}}_\Omega$ is stored in variable eEF. However this time the outer cycle is over variable a in arbitrary order (line 32) and inner cycle is over variable b in non-decreasing order[2] of lct_b (line 36). For each a we start with the

[1] Thanks to Armin Wolf for pointing out that condition $\mathrm{est}_a < \mathrm{lct}_b$ on line 7 was missing in the published version of this paper.

[2] Thanks to Joseph Scott and Roger Kameugne for correcting a mistake in the ordering that was in the published version of this paper.

Algorithm 2. Adjustments of est_i

```
1  for i ∈ T^EF do
2    est'_i := est_i ;
3  for b ∈ T^EF do begin
4    // Cases "Inside" and "Right"
5    eEF := 0;
6    ι := −1;
7    for a ∈ T^EF such that est_a < lct_b, in non-increasing order by est_a do begin
8      if lct_a ≤ lct_b then
9        eEF := eEF + e^EF_a ;
10     else if ι = −1 or min(e^EF_a, c_a (lct_b − est_a)) > min(e^EF_ι, c_ι (lct_b − est_ι))
11       then ι := a;
12     reserve := C (lct_b − est_a) − eEF − (ttAfterEst [a] − ttAfterLct [b]);
13     if ι ≠ −1 and reserve < min(e^EF_ι, c_ι (lct_b − est_ι)) then
14       est'_ι := max(est'_ι, lct_b −mandatoryIn(est_a, lct_b, ι) − ⌊reserve/ c_ι⌋);
15   end;
16   // Case "Through"
17   ι := −1;
18   for a ∈ T^EF in non-decreasing order by est_a,
19       break ties by non-increasing est_a + p^EF_a
20   do begin
21     if lct_a ≤ lct_b then begin
22       reserve := C (lct_b − est_a) − eEF − (ttAfterEst [a] − ttAfterLct [b]);
23       if ι ≠ −1 and reserve < c_ι(lct_b − est_a) then
24         est'_ι := max(est'_ι, lct_b −mandatoryIn(est_a, lct_b, ι) − ⌊reserve/ c_ι⌋);
25       eEF := eEF − e^EF_a ;
26     end;
27     if est_a + p^EF_a ≥ lct_b and (ι = −1 or c_a > c_ι) then
28       ι := a;
29   end;
30 end;
31 // Case "Left"
32 for a ∈ T^EF do begin
33   eEF := 0;
34   ι := −1;
35   Q := queue of activities i ∈ T^EF sorted by non-decreasing est_i + p^EF_i ;
36   for b ∈ T^EF in non-decreasing order by lct_b do
37     if est_a ≤ est_b then begin
38       eEF := eEF + e^EF_b ;
39       while est_Q.top + p^EF_Q.top < lct_b do begin
40         i := Q.dequeue;
41         if est_i < est_a and est_a < est_i + p^EF_i and
42             ( ι = −1 or c_i(est_i + p^EF_i − est_a) > c_ι(est_ι + p^EF_ι − est_a) )
43           then ι := i;
44       end;
45       reserve := C (lct_b − est_a) − eEF − (ttAfterEst [a] − ttAfterLct [b]);
46       if ι ≠ −1 and reserve < c_ι(est_ι + p^EF_ι − est_a) then
47         est'_ι := max(est'_ι, lct_b −mandatoryIn(est_a, lct_b, ι) − ⌊reserve/ c_ι⌋);
48     end;
49 end;
50 for i ∈ T^EF do
51   est_i := est'_i;
```

empty set Ω (line 33) and we gradually add into Ω activities b (line 38) such that $\mathrm{est}_a \leq \mathrm{est}_b$.

Activities which are in the Left position with Ω are $I = \{i,\ i \in T^{\mathrm{EF}}\ \&\ \mathrm{est}_i < \mathrm{est}_\Omega < \mathrm{est}_i + \mathrm{p}_i^{\mathrm{EF}} < \mathrm{lct}_\Omega\}$. As we iterate over b, value $\mathrm{lct}_\Omega = \mathrm{lct}_b$ is increasing and set I is growing: there are more activities i fulfilling condition $\mathrm{est}_i + \mathrm{p}_i^{\mathrm{EF}} < \mathrm{lct}_\Omega$. To enumerate all activities i as they are added into I we create a queue Q of all activities sorted by $\mathrm{est}_i + \mathrm{p}_i^{\mathrm{EF}}$ (line 35). Each time we change lct_Ω by adding b into Ω we also enumerate all activities i which newly fulfill condition $\mathrm{est}_i + \mathrm{p}_i^{\mathrm{EF}} < \mathrm{lct}_\Omega$ (lines 39–40). These activities are candidates to be added into I. If they really belong to I (line 41) then we check whether i is better than ι (line 42). Note that for position Left $\mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i) = \mathrm{c}_i(\mathrm{est}_i + \mathrm{p}_i^{\mathrm{EF}} - \mathrm{est}_\Omega)$, see Table 1. Finally, after each addition of b into Ω we check whether ι can be updated by Ω (lines 45–47).

2.5 Time Complexity

Time complexity of Algorithm 2 is $\mathcal{O}(n^2)$: there are nested cycles over variables a and b with max n iterations each. The cycle on lines 39–44 is executed at most n times for each a because each time i is removed from queue Q.

As explained before, the Algorithm 2 does not make all updates by the propagation rule in one run because it updates only activity ι with the maximum potential overload. The remaining activities are updated in the next run(s). Furthermore it does not look over all subsets $\Omega' \subseteq \Omega$ to find the best possible update as standard edge finding algorithm does. Therefore it is not possible to directly compare time complexity $\mathcal{O}(n^2)$ of Algorithm 2 with time complexities $\mathcal{O}(kn^2)$ and $\mathcal{O}(kn \log n)$ of algorithms [5, 11].

Nevertheless, we believe that additional iterations needed to reach the fixpoint are not such a big disadvantage. There are several important aspects:

1. Even if the algorithm would fix all potential overflows with the current bounds, new bounds may generate new potential overflows (especially after applying symmetrical algorithm to update lct_i). Therefore such an algorithm would not be idempotent anyway – it would be still necessary to repeat the algorithm until the fixpoint is found.
2. In practice most of the time the edge finding algorithm runs without changing any bound. For benchmarks in section 5, the algorithm changes some est_i only in circa 30% of cases. Therefore it pays off to tune the algorithm for the case where it does not propagate. Probability of two updates in two consecutive runs is only $30\% \times 30\% = 9\%$. Maybe we can save some of these 9% of runs, but it could slow down of the remaining 91% of runs.
3. For future research, it may be interesting to look for an algorithm with output-sensitive time complexity such as $\mathcal{O}(n^2 + ln)$ where l is the number of changes done.
4. This is not the first propagation algorithm which using this "lazy" approach, see for example the not-first/not-last algorithm in [10].

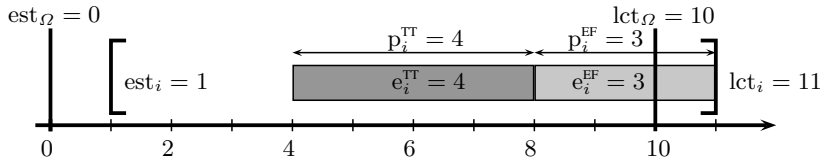


Fig. 7. An example for better counting of energy consumption in $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$.

2.6 Symmetry

Algorithm 2 updates values est_i. An algorithm to update lct_i is symmetrical. The symmetry is to consider time going backwards. It is possible to use the algorithm for est_i to do updates also on lct_i, the idea is to feed the algorithm with symmetrical data about activities:

$$\mathrm{inputEst}_i = -\mathrm{lct}_i \qquad \mathrm{inputLct}_i = -\mathrm{est}_i$$

The resulting bounds have to be also interpreted symmetrically.

3 Improvements

This section describes further improvements of algorithms 1 and 2 by incorporating some ideas from energetic reasoning and not-first/not-last.

3.1 Improved Time Bound Adjustments

As explained before, est_i can be updated if scheduling i at current est_i would cause overload in some interval $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$. In this case there are some activities which must be finished in $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$ before i can start. In particular, at least one of these activities must end before i can start. So the new est_i must be bigger than minimum $\mathrm{est}_j + \mathrm{p}_j$ over all activities j which contributes to reserve(Ω). This new lower bound for est_i can be sometimes better than the one computed by the Algorithm 2. This idea is similar to the not-first propagation rule [7].

Note that the activity which ends before i in $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$ does not have to be from set Ω. It could be also one of the activities which contributes to timetable in $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$. Furthermore, in the following section we will consider even more activities in the computation of reserve(Ω).

As activities j come from different sources, computation of minimum $\mathrm{est}_j + \mathrm{p}_j$ can slow down the algorithm. Simpler but less accurate alternative is to precompute for each activity a minimum $\mathrm{est}_j + \mathrm{p}_j$ over all $j \in T$ such that $\mathrm{est}_a < \mathrm{est}_j + \mathrm{p}_j$. Then each time we compute est'_i we can use this precomputed value (for current a) as another lower bound. This approach is used for experimental results reported at the end of this paper.

3.2 Improved Computation of Energy Consumption

Let us consider an example from Figure 7. What we see there is set Ω with $\mathrm{est}_\Omega = 0, \mathrm{lct}_\Omega = 10$ and activity $i \notin \Omega$ with $\mathrm{est}_i = 1, \mathrm{lct}_i = 11, \mathrm{p}_i = 7$ and $\mathrm{c}_i = 1$.

Algorithm 3. Improved Overload Checking

```
1  for b ∈ T^EF do begin
2    eEF := 0;
3    for a ∈ T^EF in non-increasing order by est_a do begin
4      if lct_a ≤ lct_b then
5        eEF := eEF + e_a^EF;
6      else
7        eEF := eEF + c_a max(0, lct_b − (lct_a − p_a^EF));
8      if C(lct_b − est_a) < eEF + ttAfterEst[a] − ttAfterLct[b] then
9        fail;
10   end;
11 end;
```

Activity i has both fixed and free parts: $\mathrm{p}_i^{\mathrm{TT}} = 4$, $\mathrm{e}_i^{\mathrm{TT}} = 4$, $\mathrm{p}_i^{\mathrm{EF}} = 3$, $\mathrm{e}_i^{\mathrm{EF}} = 3$. Timetable edge finding knows from timetable that at least $\mathrm{e}_i^{\mathrm{TT}} = 4$ energy units of i must be executed during $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$. However as we can see in Figure 7 there are at least 6 energy units of i which must be executed during $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$. So in this example timetable edge finding does better energy computation than standard edge finding (4 energy units versus 0) however it is still less accurate than energetic reasoning (6 energy units). In this section we show how to improve energy computation in timetable edge finding to get also 6 energy units. It is a step towards energetic reasoning, however only in limited cases.

Let's generalize the situation from Figure 7. If i is in Right or Inside position with Ω then we can further increase energy consumption in $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$ by:

$$\mathrm{sureIn}(\Omega, i) = \mathrm{c}_i \max\left(0, \mathrm{lct}_\Omega - (\mathrm{lct}_i - \mathrm{p}_i^{\mathrm{EF}})\right)$$

Algorithm 3 shows how to take $\mathrm{sureIn}(\Omega, i)$ into account in the overload checking algorithm. When activity a is not added into Ω then it is in Inside or Right position with all future sets Ω with $\mathrm{lct}_\Omega = \mathrm{lct}_b$. As $\mathrm{sureIn}(\Omega, i)$ does not depend on est_Ω we can add $\mathrm{sureIn}(\Omega, a)$ to variable eEF (line 7).

Improved energy consumption can be also taken into account in Algorithm 2. We do not present the modified algorithm in this paper. The general idea is to first store improved values eEF for all pairs of activities a and b in a two dimensional array. Then we can use this array in the update algorithm instead of eEF variable (be careful about duplicates in est_a and lct_b). To avoid counting energy $\mathrm{sureIn}(\Omega, i)$ twice for positions Inside and Right, it is necessary to decrease $\mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i)$ by $\mathrm{sureIn}(\Omega, i)$ (lines 10 and 13) and also increase $\mathrm{mandatoryIn}(\mathrm{est}_a, \mathrm{lct}_b, \iota)$ by $\mathrm{sureIn}(\Omega, i)$ at line 14.

4 Comparison with Standard and Extended Edge Finding

This section compares propagation power of timetable edge finding (without improvements from Section 3) with standard and extended edge finding algorithms.

Standard and extended edge finding propagation rules have two parts. First part considers a set $\Omega \subset T$ and an activity $i \in T \setminus \Omega$. If one of the following two

conditions holds:

edge finding (EF):	extended edge finding (EEF):
$C\left(\mathrm{lct}_\Omega - \mathrm{est}_{\Omega\cup\{i\}}\right) < \mathrm{e}_\Omega + \mathrm{e}_i$	$\mathrm{est}_i < \mathrm{est}_\Omega < \mathrm{est}_i + \mathrm{p}_i \quad \& \quad C\left(\mathrm{lct}_\Omega - \mathrm{est}_\Omega\right) < \mathrm{e}_\Omega + \mathrm{c}_i\left(\mathrm{est}_i + \mathrm{p}_i - \mathrm{est}_\Omega\right)$

then i must end after lct_Ω (i.e. $\mathrm{est}_i + \mathrm{p}_i > \mathrm{lct}_\Omega$) otherwise there is no solution. However both algorithms search for a better update of est_i. In particular they enumerate all subsets Ω' of Ω to find the best possible update:

$$\mathrm{est}_i := \max\left(\mathrm{est}_i, \max_{\substack{\Omega'\subseteq\Omega \\ \mathrm{rest}(\Omega',\mathrm{c}_i)>0}} \mathrm{est}_{\Omega'} + \left\lceil \frac{\mathrm{rest}\left(\Omega', \mathrm{c}_i\right)}{\mathrm{c}_i} \right\rceil\right) \tag{UPD}$$

where $\mathrm{rest}\left(\Omega', \mathrm{c}_i\right) = \mathrm{e}_{\Omega'} - (C - \mathrm{c}_i)\left(\mathrm{lct}_{\Omega'} - \mathrm{est}_{\Omega'}\right)$

To compare the filtering power of the rules above with timetable edge finding it is best to concentrate on fixpoints[3]. In the following we prove that when timetable edge finding reaches a fixpoint then neither standard nor extended edge finding can propagate anything. We begin by the following lemma:

Lemma 1. *If (EF) or (EEF) holds and timetable edge finding reached a fixpoint (i.e. it cannot propagate any more) then* $\mathrm{est}_i + \mathrm{p}_i > \mathrm{lct}_\Omega$.

Proof. By contradiction: we will assume that $\mathrm{est}_i + \mathrm{p}_i \leq \mathrm{lct}_\Omega$. There are 3 cases:

1. **(EEF) holds.** In this case $\mathrm{est}_i < \mathrm{est}_\Omega < \mathrm{est}_i + \mathrm{p}_i \leq \mathrm{lct}_\Omega$. Let us consider total energy used by i and Ω in interval $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$ if i is scheduled on est_i. Without timetable we can estimate it as:

   $$\mathrm{e}_\Omega + \mathrm{c}_i(\mathrm{est}_i + \mathrm{p}_i - \mathrm{est}_\Omega)$$

   With timetable we can make a better estimation:

   $$\mathrm{e}^{\mathrm{EF}}_\Omega + \mathrm{ttAfterEst}(\Omega) - \mathrm{ttAfterLct}(\Omega) + \mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i)$$

   Computation with timetable is more precise therefore:

   $$\mathrm{e}_\Omega + \mathrm{c}_i(\mathrm{est}_i + \mathrm{p}_i - \mathrm{est}_\Omega) \leq \\ \leq \mathrm{e}^{\mathrm{EF}}_\Omega + \mathrm{ttAfterEst}(\Omega) - \mathrm{ttAfterLct}(\Omega) + \mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i)$$

   This together with (EEF) results in $\mathrm{reserve}(\Omega) < \mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i)$. That contradicts the assumption that timetable edge finding cannot propagate.

[3] Note that rules (EF) and (EEF) together are monotonic, therefore by Domain Reduction Theorem [2] their repetitive application (in arbitrary order) leads to a unique fixpoint. Similarly, propagation rule behind timetable edge finding algorithm (without improvements from Section 3) is also monotonic, therefore its repetitive application also leads to a unique fixpoint.

2. **(EF) holds and $\mathbf{est_i \geq est_\Omega}$.** In this case if i is scheduled on est_i then its energy contribution to $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$ is e_i (remember that we are assuming for contradiction that $\mathrm{est}_i + \mathrm{p}_i \leq \mathrm{lct}_\Omega$). Again, this contribution is counted in timetable and in $\mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i)$. Therefore:

$$\mathrm{e}_i \leq \mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i) + \mathrm{ttAfterEst}(\Omega) - \mathrm{ttAfterLct}(\Omega)$$

   Together with (EF) it gives the contradiction $\mathrm{reserve}(\Omega) < \mathrm{add}(\mathrm{est}_\Omega, \mathrm{lct}_\Omega, i)$.
3. **(EF) holds and $\mathbf{est_i < est_\Omega}$.** Then condition (EF) can be rewritten as:

$$\begin{aligned}
C\,(\mathrm{lct}_\Omega - \mathrm{est}_i) &< \mathrm{e}_\Omega + \mathrm{e}_i && \text{because } \mathrm{est}_i < \mathrm{est}_\Omega \\
C\,(\mathrm{lct}_\Omega - \mathrm{est}_\Omega) &< \mathrm{e}_\Omega + \mathrm{e}_i - C\,(\mathrm{est}_\Omega - \mathrm{est}_i) && \\
C\,(\mathrm{lct}_\Omega - \mathrm{est}_\Omega) &< \mathrm{e}_\Omega + \mathrm{e}_i - \mathrm{c}_i\,(\mathrm{est}_\Omega - \mathrm{est}_i) && \text{because } C > \mathrm{c}_i \\
C\,(\mathrm{lct}_\Omega - \mathrm{est}_\Omega) &< \mathrm{e}_\Omega + \mathrm{c}_i\,(\mathrm{est}_i + \mathrm{p}_i - \mathrm{est}_\Omega) && \text{because } \mathrm{e}_i = \mathrm{c}_i\,\mathrm{p}_i
\end{aligned}$$

   When i is scheduled on est_i then its energy contribution to $[\mathrm{est}_\Omega, \mathrm{lct}_\Omega]$ is $\mathrm{c}_i\,(\mathrm{est}_i + \mathrm{p}_i - \mathrm{est}_\Omega)$. Together with the inequality above it means that timetable edge finding propagates what is a contradiction. □

Proposition 1. *When timetable edge finding reaches a fixpoint then both standard and extended edge finding cannot propagate anything.*

Proof. By contradiction: we will assume that timetable edge finding reached a fixpoint, i.e. there is no potential overload, however standard or extended edge finding can propagate, i.e. there is i, Ω and Ω', $\Omega' \subseteq \Omega$, such that (EF) or (EEF) holds and (UPD) improves est_i using Ω'.

We are going to prove (for the contradiction) that timetable edge finding propagates for Ω' and i. By Lemma 1 we know that $\mathrm{lct}_\Omega < \mathrm{est}_i + \mathrm{p}_i$. And because Ω' is a subset of Ω we conclude $\mathrm{lct}_{\Omega'} \leq \mathrm{lct}_\Omega < \mathrm{est}_i + \mathrm{p}_i$. We distinguish two cases:

1. **$\mathbf{est_i \leq est_{\Omega'}}$.** In this case when i is scheduled on est_i then its energy contribution to $[\mathrm{est}_{\Omega'}, \mathrm{lct}_{\Omega'}]$ is $\mathrm{c}_i\,(\mathrm{lct}_{\Omega'} - \mathrm{est}_{\Omega'})$. However because Ω' is used by (UPD) to improve est_i it must hold:

$$\begin{aligned}
0 &< \mathrm{rest}(\Omega', \mathrm{c}_i) \\
0 &< \mathrm{e}_{\Omega'} - (C - \mathrm{c}_i)\,(\mathrm{lct}_{\Omega'} - \mathrm{est}_{\Omega'}) \\
C\,(\mathrm{lct}_{\Omega'} - \mathrm{est}_{\Omega'}) &< \mathrm{e}_{\Omega'} + \mathrm{c}_i\,(\mathrm{lct}_{\Omega'} - \mathrm{est}_{\Omega'})
\end{aligned}$$

   Therefore timetable edge finding propagates what is a contradiction.

2. $\mathbf{est_i > est_{\Omega'}}$**.** In this case if i is scheduled on est_i then its energy contribution to $[\text{est}_{\Omega'}, \text{lct}_{\Omega'}]$ is $\text{c}_i\,(\text{lct}_{\Omega'} - \text{est}_i)$. However because (UPD) improves est_i:

$$\begin{aligned}
\text{est}_i &< \text{est}_{\Omega'} + \left\lceil \frac{\text{rest}(\Omega', \text{c}_i)}{\text{c}_i} \right\rceil \\
\text{est}_i &< \text{est}_{\Omega'} + \left\lceil \frac{\text{e}_{\Omega'} - (C - \text{c}_i)\,(\text{lct}_{\Omega'} - \text{est}_{\Omega'})}{\text{c}_i} \right\rceil \\
0 &< \left\lceil \frac{\text{c}_i(\text{est}_{\Omega'} - \text{est}_i) + \text{e}_{\Omega'} - (C - \text{c}_i)\,(\text{lct}_{\Omega'} - \text{est}_{\Omega'})}{\text{c}_i} \right\rceil \\
0 &< \text{c}_i(\text{est}_{\Omega'} - \text{est}_i) + \text{e}_{\Omega'} - (C - \text{c}_i)\,(\text{lct}_{\Omega'} - \text{est}_{\Omega'}) \\
C\,(\text{lct}_{\Omega'} - \text{est}_{\Omega'}) &< \text{e}_{\Omega'} + \text{c}_i\,(\text{lct}_{\Omega'} - \text{est}_i)
\end{aligned}$$

Therefore timetable edge finding propagates what is a contradiction. □

5 Experimental Results

The presented algorithm (including the improvements described in Section 3) was tested on 438 open instances of the RCPSP problem from PSPLIB [1]. The RCPSP problem is to find the shortest possible schedule for a set of activities while fulfilling precedence constraints and cumulative resource constraints. For the open instances the minimum possible length l of the schedule is still not known, but PSPLIB keeps track of the best published lower and upper bounds (LB and UB such that $\text{LB} \leq l \leq \text{UB}$), including recent results of Schutt et al. [8, 9]. PSPLIB does not contain results of Laborie [4], however these are with a single exception (lower bound 83 for instance j90_25_5) overcome by results of Schutt et al.

We improved current best known lower bounds using destructive lower bounds: first we try to find a solution with length equal to the current best known LB. We used simple SetTimes search as recapped in [3] and after each decision we propagate all constraints to fixpoint (using timetable edge finding for cumulative resources). If there is no solution then we continue by trying to find a solution with length $\text{LB} + 1$. If that also fails then we try $\text{LB} + 2$ and so on. The time limit for each improvement step is 60 seconds. Experiments were done on Intel(R) Core(TM)2 Duo CPU T9400 on 2.53GHz.

Table 2 summarizes the results. Open instances are split into 3 groups by size. For each group there is the number of open instances in the group, number of instances with LB improved by one (column LB+1), by two etc. For 9 instances we were able to improve LB without starting SetTimes search, that is, propagation itself found the problem infeasible. Detailed results can be found at `http://vilim.eu/petr/cpaior2011-results.txt`.

Acknowledgment I would like to thank Philippe Laborie and Jerôme Rogerie for their help with the algorithm and with this paper. I also thank to anonymous referees for providing very useful feedback.

Size	# Instances	LB+1	LB+2	LB+3	LB+4
60	49	5	3	-	-
90	78	24	10	-	-
120	311	82	32	9	4

Table 2. Experimental results

References

[1] Project scheduling problem library. URL http://webserver.wi.tum.de/psplib.

[2] Krzysztof R. Apt. The essence of constraint propagation. *Theoretical Computer Science*, 221(1–2):179–210, 1999.

[3] Daniel Godard, Philippe Laborie, and Wim Nuijten. Randomized large neighborhood search for cumulative scheduling. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005)*, pages 81–89. AAAI, 2005.

[4] Philippe Laborie. Complete MCS-based search: Application to resource constrained project scheduling. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI*, pages 181–186. Professional Book Center, 2005.

[5] Luc Mercier and Pascal Van Hentenryck. Edge finding for cumulative scheduling. *Informs Journal of Computing*, 20(1):143–153, 2008.

[6] Claude Le Pape Philippe Baptiste and Wim Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers, 2001.

[7] Andreas Schutt, Armin Wolf, and Gunnar Schrader. 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*, pages 66–80. Springer, 2005.

[8] Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, and Mark G. Wallace. Why cumulative decomposition is not as bad as it sounds. In *CP'09: Proceedings of the 15th international conference on Principles and practice of constraint programming*, pages 746–761. Springer-Verlag, 2009.

[9] Andreas Schutt, Thibaut Feydy, Peter Stuckey, and Mark Wallace. Explaining the cumulative propagator. *Constraints*, pages 1–33–33, 2010. ISSN 1383-7133. doi: 10.1007/s10601-010-9103-2.

[10] Philippe Torres and Pierre Lopez. On not-first/not-last conditions in disjunctive scheduling. *European Journal of Operational Research*, 127(2):332–343, 1999.

[11] Petr Vilím. Edge finding filtering algorithm for discrete cumulative resources in $O(kn \log n)$. In *CP'09: Proceedings of the 15th international conference on Principles and practice of constraint programming*, pages 802–816. Springer-Verlag, 2009.