Constraints

Overload Checking and Edge-Finding for Robust Cumulative Scheduling

--Manuscript Draft--

Manuscript Number:	CONS-D-16-00037
Full Title:	Overload Checking and Edge-Finding for Robust Cumulative Scheduling
Article Type:	Original Research
Keywords:	Constraint programming, scheduling, robustness, global constraint, filtering algorithms
Abstract:	We present two new filtering algorithms for the FLEXC constraint, a constraint that models cumulative scheduling problems where up to r tasks can be delayed while keeping the schedule valid. We adapt the overload checking and Edge-Finding filtering rules for this framework. It turns out that the complexities of the state of the art algorithms for these techniques are maintained, when the number of delayed tasks r is constant. The experimental results verify a stronger filtering for these methods when used in conjunction with time-tabling. Furthermore, the computation times show a faster filtering for specific heuristics for many instances.

Noname manuscript No.
(will be inserted by the editor)

Overload Checking and Edge-Finding for Robust Cumulative Scheduling

Hamed Fahimi · Claude-Guy Quimper

Received: date / Accepted: date

Abstract We present two new filtering algorithms for the FLEXC constraint, a constraint that models cumulative scheduling problems where up to r tasks can be delayed while keeping the schedule valid. We adapt the overload checking and Edge-Finding filtering rules for this framework. It turns out that the complexities of the state of the art algorithms for these techniques are maintained, when the number of delayed tasks r is constant. The experimental results verify a stronger filtering for these methods when used in conjunction with time-tabling. Furthermore, the computation times show a faster filtering for specific heuristics for many instances.

1 Introduction

Project scheduling problems deal with allocating scarce resources over time in order to perform jobs. This context arises in a great variety of environments, such as in the industry to plan operations. *Constraint programming* is a powerful methodology to solve large scale and practical scheduling problems. Filtering techniques are being developed and improved over the past years in constraint-based scheduling. The prominency of filtering algorithms lies on their power to shrink the search tree by excluding values from the domains which do not yield a feasible solution. *Robust project scheduling* is concerned with unruly environments which are subject to disruptive behaviours that are caused by undesirable factors. In such contexts, the execution of tasks takes longer than expected. Inevitably, re-planning is necessary, however we intend to focus on robust scheduling.

Claude-Guy Quimper
Tel.: +1-418-656-2131 ext. 2099
Fax: -
E-mail: Claude-Guy.Quimper@ift.ulaval.ca

Hamed Fahimi
Tel: +1-418-656-2131 ext. 4799
E-mail: hamed.fahimi.1@ulaval.ca

This work is based on the framework provided by Derrien et al. [7] for robust cumulative scheduling, where they assume that for a set of n tasks $\mathcal{I}$, at most r of them can be delayed without the requirement to reschedule the alternative tasks. The considered stochastic framework can capture practical problems, such as the crane assignment problem [7]. In this problem, the planner needs a schedule which still respects the deadlines in case at most r tasks are delayed. Derrien et.al present the adaption of time-tabling algorithm. Even though they define the paradigm for r delayed tasks, they focus on the case $r = 1$. This paper adapts the overload checking [20] and Edge-Finding [19] for any $r > 0$ with this new framework.

The paper is structured as follows. Section 2 describes the terms and notations which are most frequently used throughout the paper. It also presents the CUMULATIVE constraint as well as the FLEXC constraint as a variation of CUMULATIVE constraint in robust contexts. Moreover, it surveys the filtering techniques time-tabling, overload checking as well as Edge-Finding. Section 3 presents the robust algorithm of overload checking. Section 4 is devoted to the Edge-Finding algorithm for filtering the domains of the starting time variables. Section 5 presents and discuss the experiments. In section 6 we conclude.

2 Preliminaries and the general framework

We tackle the scheduling problem for a set of tasks $\mathcal{I} = \{1, ..., n\}$ to be executed on a resource of finite capacity $\mathcal{C}$. Presuming the time is discrete, that is the values of the attributes are integers, every task $i \in \mathcal{I}$ is characterized by an earliest starting time est_i, a latest completion time lct_i^0, a capacity c_i and a processing time p_i^0. The *latest starting time* (lst_i) of a task is the largest date at which it can start executing and the *earliest completion time* (ect_i) of a task is the earliest date at which it can cease to execute. These two values are computed with $\mathrm{lst}_i = \mathrm{lct}_i^0 - p_i^0$ and $\mathrm{ect}_i = \mathrm{est}_i + p_i^0$, respectively. The *starting time* S_i of a task is the time point at which it starts executing. The starting times are the unknowns of a scheduling problem. The *energy* of i, denoted e_i, is the sum over time of the capacity of i, computed by $\mathrm{e}_i = \mathrm{c}_i \, p_i^0$. One may generalize these notions for an arbitrary subset $\Theta \subseteq \mathcal{I}$ of tasks as follows

$$\mathrm{est}_\Theta = \min\{\mathrm{est}_i : i \in \Theta\} \tag{1}$$

$$\mathrm{lct}_\Theta^0 = \max\{\mathrm{lct}_i^0 : i \in \Theta\} \tag{2}$$

$$\mathrm{e}_\Theta = \sum_{i \in \Theta} \mathrm{e}_i \tag{3}$$

For an empty set we assume that $\mathrm{est}_\emptyset = \infty$ and $\mathrm{lct}_\emptyset^0 = -\infty$.

Vilím [19] introduces the concept of the *earliest energy envelope* and the *latest energy envelope* of Θ, which are respectively defined by

$$\mathrm{Env}_\Theta = \max_{\Omega \subseteq \Theta} (\mathcal{C} \, \mathrm{est}_\Omega + \mathrm{e}_\Omega) \tag{4}$$

$$\mathrm{Env}'_\Theta = \min_{\Omega \subseteq \Theta} (\mathcal{C} \, \mathrm{lct}_\Omega^0 - \mathrm{e}_\Omega) \tag{5}$$

For $\Omega \subseteq \Theta$, Env_Θ provides the aggregation of the energy which is required to fully use the resource up to time est_Ω and the energy of executing the task in Ω. Note that a subset $\Omega \subset \Theta$ might give the maximum, if $\mathrm{est}_\Omega > \mathrm{est}_\Theta$. A similar reasoning for Env'_Θ holds.

It is convenient to compute the latest starting time and earliest completion time of a set of tasks Θ. Thanks to (4) and (5), a lower bound for the earliest completion time and an upper bound for the latest starting time of Θ are respectively obtained by $\mathrm{ect}_\Theta = \lceil \mathrm{Env}_\Theta / \mathcal{C} \rceil$ and $\mathrm{lst}_\Theta = \lfloor \mathrm{Env}'_\Theta / \mathcal{C} \rfloor$ [19].

Due to some considerable uncertainties, one can not absolutely trust the task durations, as the processing of tasks can take longer than expected. Such a situation entails the assignment of an attribute d_i known as the *delay duration* to each task i. From that, a task i has two extra parameters, the delayed processing time $p_i^1 = p_i^0 + d_i$ and the delayed latest completion time $\mathrm{lct}_i^1 = \mathrm{lct}_i^0 + d_i$. Furthermore,

$$\mathrm{lct}_\Theta^1 = \max\{\mathrm{lct}_i^1 \mid i \in \Theta\} \tag{6}$$

We refer to the tasks when they are delayed or not delayed with particular symbols. A task i that is not delayed is called *regular* and it is denoted i^0 and a task i that is delayed is denoted i^1. Throughout this paper, we associate a regular task with 0 and a delayed task with 1. For instance, the set $\{A^0, B^1, C^0, D^1\}$ is a set of tasks where A and C are not delayed and B and D are delayed. $\mathcal{I}^0 = \{i^0 \mid i \in \mathcal{I}\}$ refers to the regular tasks from $\mathcal{I}$ and $\mathcal{I}^1 = \{i^1 \mid i \in \mathcal{I}\}$ refers to the delayed tasks from $\mathcal{I}$. Moreover, for $i^1 \in \mathcal{I}^1$, the processing time p_i^1 is considered and for $i^0 \in \mathcal{I}^0$, the processing time p_i^0 is considered. For $i \in \mathcal{I}$ and $b \in \{0, 1\}$, we define $\mathrm{e}_{i^b} = p_i^b \, \mathrm{c}_i$. For a subset $\Theta \subseteq \mathcal{I}^0 \cup \mathcal{I}^1$, we define

$$\mathrm{e}_\Theta = \sum_{i^b \in \Theta} \mathrm{e}_{i^b} \tag{7}$$

For a subset $\Theta \subseteq \mathcal{I} \cup \mathcal{I}^0 \cup \mathcal{I}^1$,

$$\mathcal{I}(\Theta) = \{i \in \mathcal{I} \mid i^0 \in \Theta \vee i^1 \in \Theta \vee i \in \Theta\} \subseteq \mathcal{I} \tag{8}$$

Note that $\mathcal{I}(\Theta)$ is the set of tasks from $\mathcal{I}$, whatsoever. That is, the delay status of the tasks is not specified in $\mathcal{I}(\Theta)$.

2.1 Cumulative constraint

The Cumulative constraint models a relationship between a scarce resource and the tasks which are to be processed on the resource. Let $S_1, ..., S_n$ denote the starting times of the tasks as the decision variables of the problem, where the domain of $S_i, i \in \mathcal{I}$, is the interval $[\mathrm{est}_i, \mathrm{lst}_i]$. The constraint Cumulative$([S_1, \ldots, S_n], [p_1^0, \ldots, p_n^0], [\mathrm{c}_1, \ldots, \mathrm{c}_n], \mathcal{C})$ holds if and only if

$$\forall t : \sum_{S_i \le t < S_i + p_i^0} \mathrm{c}_i \le \mathcal{C} \tag{9}$$

The decision problem related to the Cumulative constraint is called the *cumulative scheduling problem* (CuSP) [1].

2.2 Robust cumulative constraint

Derrien et.al [7] introduce the Robust Cumulative Problem of order r (RCuSP^r) by integrating the notion of robustness to CuSP. According to this framework, a set $\Theta^1 \subseteq \mathcal{I}^1$ of at most $r \geq 1$ tasks can be delayed up to their associated delay attribute without shifting the position of other tasks. A solution to RCuSP^r satisfies

$$\forall t, \forall \Theta^1 \subseteq \mathcal{I}^1, |\Theta^1| \leq r : \sum_{i \in \mathcal{I}: S_i \leq t < S_i + p_i^0} \mathrm{c}_i + \sum_{j^1 \in \Theta^1 : S_j + p_j^0 \leq t < S_j + p_j^1} \mathrm{c}_j \leq \mathcal{C} \quad (10)$$

The first summation in constraint (10) adds the capacities of all tasks executing at time t and the second summation adds the capacities of at most r tasks whose delayed part intersects with time t. We refer to the constraint (10) by
$\textsc{FlexC}([S_1, \ldots, S_n], [p_1^0, \ldots, p_n^0], [d_1, \ldots, d_n], [\mathrm{c}_1, \ldots, \mathrm{c}_n], \mathcal{C}, r)$.
Let

$$1(x) = \begin{cases} 1 \text{ if } x \text{ is true} \\ 0 \text{ if } x \text{ is false} \end{cases} \quad (11)$$

The following relation indicates that this problem considers $\binom{n}{r}$ scenarios where r tasks among n are delayed and the CUMULATIVE constraint holds no matter which of r tasks are delayed [7].

$$\textsc{FlexC}([S_1, \ldots, S_n], [p_1^0, \ldots, p_n^0], [d_1, \ldots, d_n], [\mathrm{c}_1, \ldots, \mathrm{c}_n], \mathcal{C}, r) \iff$$

$$\bigwedge_{\substack{\Theta^1 \subseteq \mathcal{I}^1 \\ |\Theta^1| = r}} \textsc{Cumulative}([S_1, ..., S_n], [p_1^{1(1 \in \Theta^1)}, ..., p_n^{1(n \in \Theta^1)}], [\mathrm{c}_1, ..., \mathrm{c}_n], \mathcal{C}) \quad (12)$$

The algorithms that we adapt to cover delayed tasks efficiently emulates the state of the art algorithms on the conjunction of CUMULATIVE constraints (12).

2.3 Time-Tabling

Time-Tabling is a filtering technique which maintains a minimal resource consumption at each time t, which allows the solver to restrict the domains of other tasks by preventing them from execution at times that would lead to the over-consumption of the resource. For a task $i \in \mathcal{I}$, if $\mathrm{lst}_i < \mathrm{ect}_i$, the time window $[\mathrm{lst}_i, \mathrm{ect}_i)$ is called the *compulsory part* of task i. That is, if it exists, a compulsory part is a time window in which a task certainly executes. Let $f(t, \Theta) = \sum_{i \in \Theta | t \in [\mathrm{lst}_i, \mathrm{ect}_i)} c_i$ denote the amount of energy that is consumed by the tasks of $\Theta \subseteq \mathcal{I}$ for which t lies in their compulsory parts. $f(t, \Theta)$ provides a lower bound on the resource consumption at time t. The time tabling rules are as follows, where the rule (13) filters the earliest starting times and the rule (14) filters the latest completion times. The left side of the implications corresponds to the detection test and the right side corresponds to the adjustment.

$$(\mathrm{c}_i + f(t, \mathcal{I} \setminus \{i\}) > \mathcal{C}) \wedge (t < \mathrm{ect}_i) \Rightarrow \mathrm{est}_i > t \quad (13)$$

$$(\mathrm{c}_i + f(t, \mathcal{I} \setminus \{i\}) > \mathcal{C}) \wedge (\mathrm{lst}_i \leq t) \Rightarrow \mathrm{lct}_i^0 < t - p_i \quad (14)$$

Notice that after applying the rules, the same task might get filtered further with respect to another time t.

Several algorithms apply the time-tabling rules [4,5,9,11,13,16]. Derrien et al., adapt the sweep algorithm proposed by Arnaud Letort, et.al. in [12] for the FLEXC constraint. It ensures that even if up to r tasks are delayed, no time tabling rules are triggered.

2.4 Overload checking

The *overload checking* rule ensures that the energy of a set of tasks $\Theta \subseteq \mathcal{I}$ does not overflow with the capacity of the resource available within the window where the tasks must execute. That is,

$$\forall \Theta \subseteq \mathcal{I} : \mathrm{e}_\Theta \leq C(\mathrm{lct}^0_\Theta - \mathrm{est}_\Theta) \tag{15}$$

Overload checking provides a necessary condition for the existence of a solution. Wolf and Schrader introduced an algorithm for overload checking which runs in $O(n\log(n))$ [21]. Vilím [20] introduced the idea of $\Theta-$tree, which is a balanced binary tree, and by taking advantage of that he also presented an algorithm in $O(n\log(n))$ for the overload checking. Recently, Fahimi and Quimper [8] proposed a linear time algorithm for this test.

2.5 Edge-Finding

This is a filtering technique which was firstly introduced by Baptiste et.al, [3]. The idea of Edge Finding for filtering the lower bounds of starting times is to detect a subset of tasks $\Theta \subset \mathcal{I}$ and a task $i \in \mathcal{I} \setminus \Theta$ such that, in any solution, all the tasks of Θ complete before i completes. This property is denoted $\Theta \prec i$ and it is called a *precedence*. The following rules capture the bounding techniques of Edge-Finding:

$$\forall \Theta, \forall i \notin \Theta, C(\mathrm{lct}^0_\Theta - \mathrm{est}_{\Theta\cup\{i\}}) < \mathrm{e}_{\Theta\cup\{i\}} \Rightarrow \Theta \prec i \tag{16}$$

$$\forall \Theta, \forall i \notin \Theta, C(\mathrm{lct}^0_{\Theta\cup\{i\}} - \mathrm{est}_\Theta) < \mathrm{e}_{\Theta\cup\{i\}} \Rightarrow i \prec \Theta \tag{17}$$

The idea can be deduced from the overload checking, in the sense that if the scheduling of task i at its release time causes an overload in the interval where Θ is allowed to execute, then Θ has to precede i. A symmetric reasoning applies for the rule (17). Once the appropriate Θ and i are identified, the temporal time bounds must be adjusted. For $\Omega \subseteq \Theta$, there are e_Ω units of energy within the interval $[\mathrm{est}_\Omega, \mathrm{lct}^0_\Omega)$. Let

$$\mathrm{rest}(\Omega, c_i) = \mathrm{e}_\Omega - (\mathcal{C} - \mathrm{c}_i)(\mathrm{lct}^0_\Omega - \mathrm{est}_\Omega) \tag{18}$$

Baptiste et al., [3] proved that the subsets which have enough energy to prevent concurrent scheduling of i are those for which $\mathrm{rest}(\Omega, c_i) > 0$ and if so, $\mathrm{est}_\Omega + \lceil \mathrm{rest}(\Omega, c_i) \rceil$

provides a lower bound for est_i and $\mathrm{lct}^0_\Omega - \lceil \mathrm{rest}(\Omega, c_i) \rceil$ provides an upper bound for lct^0_i. The following relations for the new bounds are deduced:

$$\mathrm{est}_i \leftarrow \max(\mathrm{est}_i, \max_{\substack{\emptyset \neq \Omega \subseteq \Theta \\ \mathrm{e}_\Omega > (\mathcal{C} - \mathrm{c}_i)(\mathrm{lct}^0_\Omega - \mathrm{est}_\Omega)}} \{\mathrm{est}_\Omega + \left\lceil \frac{\mathrm{rest}(\Omega, c_i)}{\mathrm{c}_i} \right\rceil\}) \tag{19}$$

$$\mathrm{lct}^0_i \leftarrow \min(\mathrm{lct}^0_i, \min_{\substack{\emptyset \neq \Omega \subseteq \Theta \\ \mathrm{e}_\Omega > (\mathcal{C} - \mathrm{c}_i)(\mathrm{lct}^0_\Omega - \mathrm{est}_\Omega)}} \{\mathrm{lct}^0_\Omega - \left\lceil \frac{\mathrm{rest}(\Omega, c_i)}{\mathrm{c}_i} \right\rceil\}) \tag{20}$$

Several algorithms exists for Edge-Finding [15,14]. Vilím introduces an algorithm which admits a running time of $O(kn \log(n))$, where k signifies the number of distinct capacities associated to the tasks in $\mathcal{I}$ [19] . Roger Kameugne et. al, [10] present an Edge-Finding algorithm in $O(n^2)$. They empirically prove that their algorithm is substantially faster than Vilím's Edge-Finding.

3 Robust overload checking

The objective of this section is to adapt the overload checking algorithm introduced in [20] for the FLEXC$([S_1, \dots, S_n], [p^0_1, \dots, p^0_n], [d_1, \dots, d_n], [\mathrm{c}_1, \dots, \mathrm{c}_n], \mathcal{C}, r)$ constraint. In section 3.1 we establish the generic form of the robust overload checking rule and illustrate it with an example. Section 3.2 introduces the notion of earliest energy envelope of a set of tasks in the robust context. Section 3.3 recasts the robust overload checking rule in terms of the robust energy envelope. Afterwards, in section 3.4 we propose an extended data structure which enables us to handle the tasks when a certain number of them are delayed. Finally, in section 3.5 we take advantage of the introduced data structure to present the robust overload checking algorithm. Moreover, we trace few steps of the algorithms for the example provided. This section finishes with a discussion on the time complexity of the algorithm.

3.1 The general form of robust overload checking rule

Let $\Theta^0 \subseteq \mathcal{I}^0, \Theta^1 \subseteq \mathcal{I}^1$, such that $|\Theta^1| \leq r$ and $\mathcal{I}(\Theta^0) \cap \mathcal{I}(\Theta^1) = \emptyset$. The overload checking triggers a failure if

$$\mathcal{C}(\max(\mathrm{lct}^0_{\mathcal{I}(\Theta^0)}, \mathrm{lct}^1_{\mathcal{I}(\Theta^1)}) - \mathrm{est}_{\mathcal{I}(\Theta^0 \cup \Theta^1)}) < \mathrm{e}_{\Theta^0} + \mathrm{e}_{\Theta^1} \tag{21}$$

Example 1 Consider the table 1 which corresponds to a set of tasks $\mathcal{I} = \{A, B, C, D, E\}$ that must execute on a resource of capacity $\mathcal{C} = 4$ in the context where at most $r = 2$ tasks are allowed to delay. Let $\Theta = \{A, B, C, D\} \subseteq \mathcal{I}$. If $\Theta^0 = \{A^0, C^0\}$ and $\Theta^1 = \{B^1, D^1\}$, the overload checking returns a failure, for

$$100 = 4(25 - 0) < (10 + 12) + (36 + 44) = 102$$

□

task	est	lct^0	p^0	d	c
A	1	20	5	1	2
B	4	23	7	2	4
C	0	14	4	0	3
D	0	21	8	3	4
E	9	26	2	1	1

Table 1 A set of tasks $\mathcal{I} = \{A, B, C, D, E\}$ to execute on a resource of capacity $\mathcal{C} = 4$. The overload checking fails according to (21) for $\Theta = \{A, B, C, D\} \subseteq \mathcal{I}, \Theta^0 = \{A^0, C^0\}$ and $\Theta^1 = \{B^1, D^1\}$.

3.2 Robust earliest energy envelope

One can generalize the notion of the earliest energy envelope when tasks can be delayed. Rather than computing the earliest energy envelope of a set Θ as in (4), we compute the earliest energy envelope of two sets $\Theta^0 \subseteq \mathcal{I}^0$ and $\Theta^1 \subseteq \mathcal{I}^1$. The tasks in Θ^0 can be regular while the tasks in Θ^1 can be delayed. The tasks that belong to both sets can either be regular or delayed but not both. The earliest energy envelope for the sets Θ^0 and Θ^1 in such a case is defined

$$\text{Env}^r(\Theta^0, \Theta^1) = \max_{\substack{\Omega^0 \subseteq \Theta^0 \\ \Omega^1 \subseteq \Theta^1 \\ |\Omega^1| \leq r \\ \mathcal{I}(\Omega^0) \cap \mathcal{I}(\Omega^1) = \emptyset}} (\mathcal{C}\, \text{est}_{\mathcal{I}(\Omega^0 \cup \Omega^1)} + \text{e}_{\Omega^0} + \text{e}_{\Omega^1}) \tag{22}$$

Once (22) is computed, a lower bound for the earliest completion time of $\Theta^0 \cup \Theta^1$ is obtained by $\lceil \text{Env}^r(\Theta^0, \Theta^1)/\mathcal{C} \rceil$.

3.3 Robust overload checking rule in terms of robust energy envelope

We establish an equivalent criterion for the overload checking rule (21) in terms of $\text{Env}^r(\Theta^0, \Theta^1)$. Let $T = \{\text{lct}_i^0 : i \in \mathcal{I}\} \cup \{\text{lct}_i^1 : i \in \mathcal{I}\}$ be the set of all latest completion time and delayed latest completion time points. For $t \in T$, let $\text{Lcut}^0(t) = \{i^0 \in \mathcal{I}^0 : \text{lct}_i^0 \leq t\}$ be the *regular left cut* of t and $\text{Lcut}^1(t) = \{i^1 \in \mathcal{I}^1 : \text{lct}_i^1 \leq t\}$ be the *delayed left cut* of t. The former signifies the set of regular tasks which terminate no later than t and the latter refers to the set of delayed tasks which terminate no later than t.

Lemma 1 *The overload checking fails according to the rule (21) if and only if for some $t \in T$*

$$\text{Env}^r(\text{Lcut}^0(\text{t}), \text{Lcut}^1(\text{t})) > \mathcal{C} \cdot \text{t}$$

Proof Consider $\Theta^0 \subseteq \mathcal{I}^0$ and $\Theta^1 \subseteq \mathcal{I}^1$ with $|\Theta^1| \leq r$ and $\mathcal{I}(\Theta^0) \cap \mathcal{I}(\Theta^1) = \emptyset$ as the subsets for which (21) implies that the overload checking fails and let $\max(\text{lct}^0_{\mathcal{I}(\Theta^0)}, \text{lct}^1_{\mathcal{I}(\Theta^1)}) = \text{lct}^0_{\mathcal{I}(\Theta^0)}$ in (21). If j corresponds to a task $j \in \mathcal{I}(\Theta^0)$ for which $\text{lct}_j^0 = \text{lct}^0_{\mathcal{I}(\Theta^0)}$ and by setting $t = \text{lct}_j^0$, the assumption for the selected task j implies $\Theta^0 \subseteq \text{Lcut}^0(t)$ and $\Theta^1 \subseteq \text{Lcut}^1(t)$. From (21) it follows that

$$\mathcal{C} \cdot t = \mathcal{C} \cdot \text{lct}^0_{\mathcal{I}(\Theta^0)} < \mathcal{C} \cdot \text{est}_{\mathcal{I}(\Theta^0 \cup \Theta^1)} + \text{e}_{\Theta^0} + \text{e}_{\Theta^1} \leq \text{Env}^r(\text{Lcut}^0(\text{t}), \text{Lcut}^1(\text{t}))$$

A similar reasoning holds if $\max(\mathrm{lct}^0_{\mathcal{I}(\Theta^0)}, \mathrm{lct}^1_{\mathcal{I}(\Theta^1)}) = \mathrm{lct}^1_{\mathcal{I}(\Theta^1)}$.

Now, assume that for some $t \in T$, $\mathrm{Env}^r(\mathrm{Lcut}^0(t), \mathrm{Lcut}^1(t)) > \mathcal{C} \cdot t$. Let $\Theta^0 \subseteq \mathrm{Lcut}^0(t)$, $\Theta^1 \subseteq \mathrm{Lcut}^1(t)$ be the subsets for which

$$\mathrm{Env}^r(\mathrm{Lcut}^0(t), \mathrm{Lcut}^1(t)) = \mathcal{C}\,\mathrm{est}_{\mathcal{I}(\Theta^0 \cup \Theta^1)} + \mathrm{e}_{\Theta^0} + \mathrm{e}_{\Theta^1}$$

where $|\Theta^1| \le r$ and $\mathcal{I}(\Theta^0) \cap \mathcal{I}(\Theta^1) = \emptyset$. Therefore,

$$\mathcal{C}\cdot\max(\mathrm{lct}^0_{\mathcal{I}(\Theta^0)}, \mathrm{lct}^1_{\mathcal{I}(\Theta^1)}) \le \mathcal{C}\cdot t < \mathrm{Env}^r(\mathrm{Lcut}^0(t), \mathrm{Lcut}^1(t)) = \mathcal{C}\,\mathrm{est}_{\mathcal{I}(\Theta^0 \cup \Theta^1)} + \mathrm{e}_{\Theta^0} + \mathrm{e}_{\Theta^1}$$

which implies (21). □

In the following, we present a data structure from which $\mathrm{Env}^r(\Theta^0, \Theta^1)$ can be retrieved efficiently.

3.4 Θ^r_L-tree

Vilím introduces the concept of $\Theta-$tree [20], which is a balanced binary tree with one leaf for each task. In the $\Theta-$tree, the leaves, when enumerated from left to right, correspond to the tasks sorted in non-decreasing order of the earliest starting times. Each node holds the parameters of energy, defined by (3) as well as the earliest energy envelope, defined by (4). For a leaf v of the $\Theta-$tree, which corresponds to a task $i \in \mathcal{I}$, the energy and the earliest energy envelope are computed for $\Theta = \{i\}$ and for an inner node w of the $\Theta-$tree, these parameters are computed for the set of all tasks which are included in the subtree rooted at w. The parameters for w are computed in a bottom-up manner. That is, the parameters of every inner node in the $\Theta-$tree depend on the parameters of its children. The structure of the tree is important in the sense that it allows to compute the earliest energy envelope by updating the tree through a bottom-up traversal to the root and ultimately retrieving the earliest completion time of the set of tasks which are scheduled thus far. While maintaining the same structure for the $\Theta-$tree, we extend it to a tree, called Θ^r_L-tree, by adding new parameters to the nodes to handle the case where at most r tasks could be delayed. In the Θ^r_L-tree, two sets of task $\Theta^0 \subseteq \mathcal{I}^0$ and $\Theta^1 \subseteq \mathcal{I}^1$ affect the values of the parameters in the leaves and therefore in the entire tree. In what follows, the symbols $\mathrm{e}^0_v, \mathrm{Env}^0_v$ and $\mathrm{e}^1_v, \mathrm{Env}^1_v$ stand for the energy and the earliest energy envelope of a task v whether this task is regular or delayed. They are computed as

$$\mathrm{e}^k_v = \begin{cases} \mathrm{c}_v\, p^0_v & \text{if } (v^0 \in \Theta^0) \wedge (k = 0 \vee v^1 \notin \Theta^1) \\ \mathrm{c}_v\, p^1_v & \text{if } (v^1 \in \Theta^1) \wedge (k > 0) \\ 0 & \text{otherwise} \end{cases} \tag{23}$$

$$\mathrm{Env}^k_v = \begin{cases} \mathcal{C}\,\mathrm{est}_v + \mathrm{e}^k_v & \text{if } (v^0 \in \Theta^0) \vee (v^1 \in \Theta^1) \\ -\infty & \text{otherwise} \end{cases} \tag{24}$$

The superscript k stands for an upper bound on the number of delayed tasks in the leaf v. Since the task corresponding to v is either regular or delayed, only two cases for $k (k \in \{0, 1\})$ make sense. However, in order to make the computations of the energy and envelope symmetrical over all nodes of the tree, we suppose that $0 \leq k \leq r$ and since e_v^k and Env_v^k are the energy and earliest energy envelope when at most k tasks are delayed, for $k \geq 2$ in the leaves of the Θ_L^r-tree we necessarily have $\mathrm{e}_v^k = \mathrm{e}_v^1$ and $\mathrm{Env}_v^k = \mathrm{Env}_v^1$.

Let w be an internal node of the tree. The symbols e_w^r and Env_w^r stand for the energy and envelope of all tasks associated to the leaves that are descendants of w. Moreover, among these tasks, a most r tasks in Θ^1 can be delayed. If the left and right children of w are respectively denoted left(w) and right(w), it is proven [19] that

$$\mathrm{e}_w^0 = \mathrm{e}_{\mathrm{left}(w)}^0 + \mathrm{e}_{\mathrm{right}(w)}^0 \tag{25}$$

$$\mathrm{Env}_w^0 = \max(\mathrm{Env}_{\mathrm{left}(w)}^0 + \mathrm{e}_{\mathrm{right}(w)}^0, \mathrm{Env}_{\mathrm{right}(w)}^0) \tag{26}$$

Scheduling a regular or delayed task is equivalent to adding the task i^0 to Θ^0 or the task i^1 to Θ^1 by updating the node corresponding to the task according to (23) and (24). Afterwards, the values of the energy and envelope for the set of all tasks which are inserted in the tree so far can be recursively computed for the internal nodes of the tree. For at most k delayed tasks the contribution of at most j delayed tasks in the left and at most $k - j$ tasks in the right subtree emanating from w, $0 \leq j \leq k$, must be added up. Hence the energies of all tasks where at most k tasks are delayed, denoted e_w^k, and the sum of the earliest energy envelope of all tasks where at most k tasks are delayed, denoted Env_w^k, $0 \leq k \leq r$, are recursively computed as

$$\mathrm{e}_w^k = \max_{0 \leq j \leq k} \{\mathrm{e}_{\mathrm{left}(w)}^j + \mathrm{e}_{\mathrm{right}(w)}^{k-j}\} \tag{27}$$

$$\mathrm{Env}_w^k = \max_{0 \leq j \leq k} \{\mathrm{Env}_{\mathrm{left}(w)}^j + \mathrm{e}_{\mathrm{right}(w)}^{k-j}\} \cup \{\mathrm{Env}_{\mathrm{right}(w)}^k, \mathrm{Env}_w^{k-1}\} \tag{28}$$

In the case that the number of delayed task is greater than the number of available nodes in the subtree of the right side, it is sufficient to retrieve Env_w^{k-1}, i.e. the energy envelope for when at most $k - 1$ tasks are delayed. At the root of the Θ_L^r-tree, we obtain $\mathrm{Env}_{\mathrm{root}}^r = \mathrm{Env}^r(\Theta^0, \Theta^1)$. This quantity is essential to perform the overload checking.

Lemma 2 *The update of the Θ_L^r-tree runs in $\Theta(r^2 \log(n))$.*

Proof Upon the update of the values e_v^k and Env_v^k of a leaf as well as all nodes along the path connecting the leaf node to the root of the Θ_L^r-tree, there are r functions e_v^k to compute in constant time. There are also r functions Env_v^k, each one computed in $O(r)$ time, which deduces $\Theta(r^2 \log(n))$ computations. □

3.5 Robust overload checking algorithm

Let t_1 and t_2 be two arbitrary time points such that $t_1 < t_2$. The overload checking test ensures that the total energy of the tasks executing within the interval $[t_1, t_2]$ does not exceed the total energy available inside it. For the rule (15) it turns out that it suffices to check it for $t_1 = \mathrm{est}_i, t_2 = \mathrm{lct}_j^0$ for $i, j \in \mathcal{I}$. In the algorithm that we propose, t_2 could also be $t_2 = \mathrm{lct}_j^1$ for $j^1 \in \mathcal{I}^1$.

Algorithm 1: Overload checking($\mathcal{I}$, $\mathcal{C}$,r)

```
1 $\Theta^0 \leftarrow \emptyset$;
2 $\Theta^1 \leftarrow \emptyset$;
3 $T \leftarrow \{\mathrm{lct}_i^0 : i \in \mathcal{I}\} \cup \{\mathrm{lct}_i^1 : i \in \mathcal{I}\}$;
4 for $t \in T$ sorted in non-decreasing order do
5     $\Theta^0 \leftarrow \Theta^0 \cup \{i \in \mathcal{I} : \mathrm{lct}_i^0 = t\}$;
6     $\Theta^1 \leftarrow \Theta^1 \cup \{i \in \mathcal{I} : \mathrm{lct}_i^1 = t\}$;
7     if $\mathrm{Env}^r(\Theta^0, \Theta^1) > \mathcal{C} \cdot t$ then
8         fail;
```

In order to develop the overload checking rule in the robust context, et $T = \{\mathrm{lct}_i^0 : i \in \mathcal{I}\} \cup \{\mathrm{lct}_i^1 : i \in \mathcal{I}\}$ be the set of all latest completion time and delayed latest completion time points. The algorithm starts with an empty Θ_L^r-tree, i.e. $\Theta^0 = \Theta^1 = \emptyset$. That is, no tasks is initially scheduled. The idea is to process the time points $t \in T$ in non-decreasing order and schedule the regular tasks whose latest completion time is equal to t by adding them to Θ^0 or schedule the delayed tasks whose delayed latest completion time is equal to t by adding them to Θ^1. Such an addition changes the energy and envelopes for the leaves corresponding to the tasks in the tree as well as for all nodes on the branch connecting the leaves to the root. At each iteration, if a task i satisfies $\mathrm{lct}_i^0 = t$, then it is scheduled on the Θ_L^r-tree and i^0 is added to Θ^0 and if $\mathrm{lct}_i^1 = t$, then i is updated in the Θ_L^r-tree and i^1 is added to Θ^1. Once scheduling the tasks corresponding to t is over, the overload checking rule (21) must be assessed. Thanks to lemma 1, in order to assess (21) in the algorithm, it suffices to check $\mathrm{Env}^r(\mathrm{Lcut}^0(\mathrm{t}), \mathrm{Lcut}^1(\mathrm{t})) > \mathcal{C} \cdot \mathrm{t}$ for $t \in T$ which is being processed and $\mathrm{Env}^r(\mathrm{Lcut}^0(\mathrm{t}), \mathrm{Lcut}^1(\mathrm{t}))$ can be retrieved from the root of the Θ_L^r-tree that is developed so far. The algorithm 1 implements the overload checking algorithm in the robust context.

In order to elucidate the mechanism of the algorithm 1, we present few steps of implementing the algorithm for the example 1 and illustrate them in figure 1. Note that the nodes of the Θ_L^r-tree that are affected during the updates that occur at each step are discriminated in blue colours. For this example we have $\mathcal{I} = \{A, B, C, D, E\}$ and $T = \{14, 20, 21, 23, 24, 25, 26, 27\}$. After initializing an empty Θ_L^r-tree, in the first iteration the regular tasks i^0 for which $\mathrm{lct}_i^0 = 14$ and the delayed tasks i^1 for which $\mathrm{lct}_i^1 = 14$ are scheduled. C is the only tasks which qualifies in both conditions. Therefore, C^0 is added to Θ^0 and C^1 is added to Θ^1. The figure 1a depicts the status of the Θ_L^r-tree after this iteration. In the next iteration, A is the only eligible task to be schedule, but it is not allowed to be delayed, as $\mathrm{lct}_A^0 = 20$ and $\mathrm{lct}_A^1 = 21 > t = 20$. Therefore A^0 is added to Θ^0 but A^1 is not yet added to Θ^1. The figure 1b illustrates the status of the Θ_L^r-tree after this iteration. In the next iteration, corresponding to the figure 1c, A^1 and D^0 are scheduled, as $\mathrm{lct}_A^1 = \mathrm{lct}_D^0 = 21$. The figure 1d demonstrates the status of the Θ_L^r-tree at iteration $t = 25$. Before $t = 25$ is processed at this step $\Theta^0 = \{C^0, A^0, D^0, B^0\}$ and $\Theta^1 = \{C^1, A^1, D^1\}$. Since $\mathrm{lct}_B^1 = 25$, B^1 must be added to Θ^1. The insertion of B^1 into the Θ_L^r-tree causes the overload checking to fail, as $\mathrm{Env}^2(\Theta^0, \Theta^1) = \mathrm{Env}^2_{\mathrm{root}} = 102 > 4 \cdot 25 = 100$.

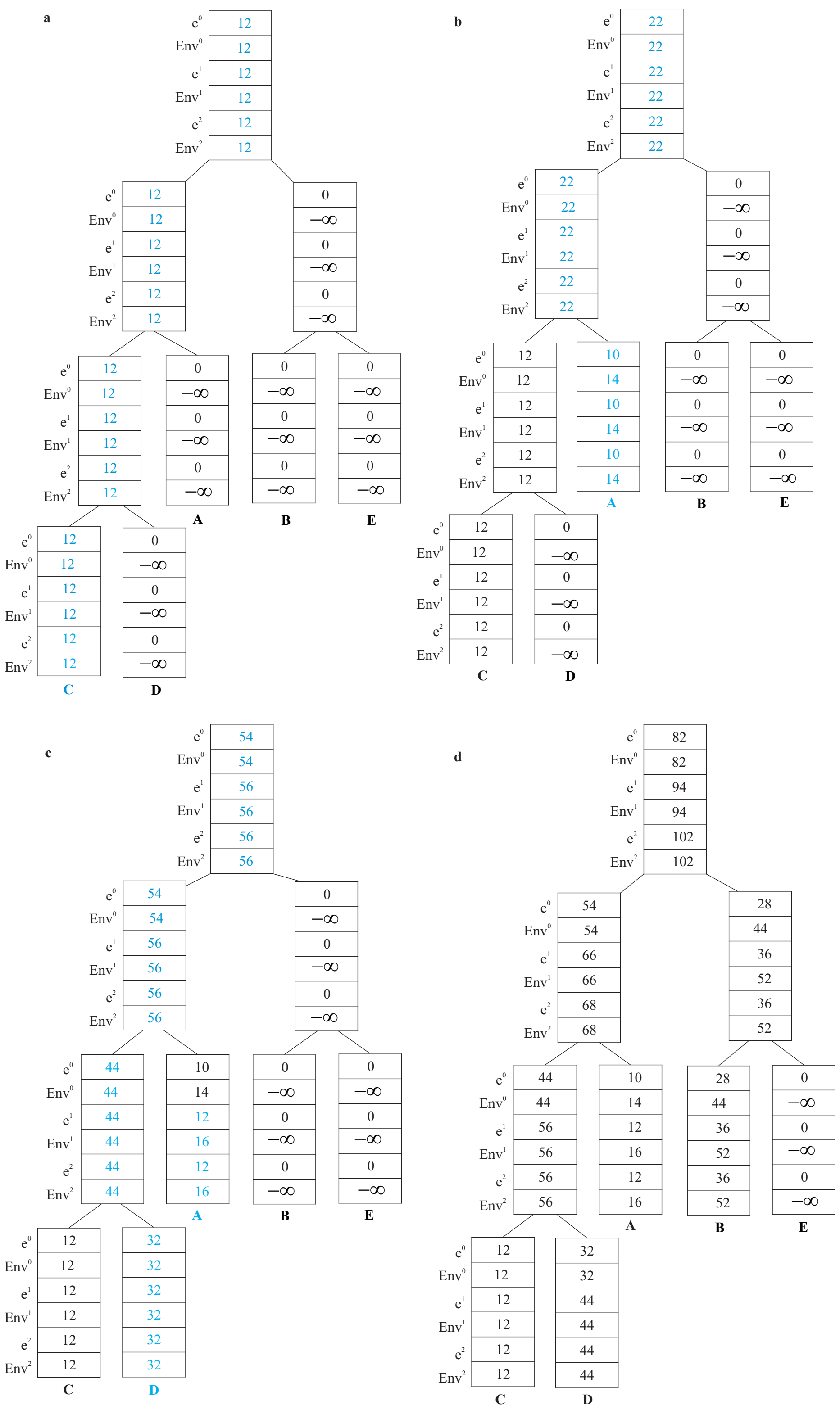


Fig. 1 In the picture (a), C^0 and C^1 are scheduled in the Θ_L^r-tree. The coloured nodes in blue represent the affected nodes during the update of the tree. In this case, $\Theta^0 = \{C^0\}$ and $\Theta^1 = \{C^1\}$. In the picture (b), A^0 is scheduled in the Θ_L^r-tree, but not A^1. In this case, $\Theta^0 = \{C^0, A^0\}$ and $\Theta^1 = \{C^1\}$. In the picture (c), A^1 and D are scheduled. In this case, $\Theta^0 = \{C^0, A^0, D^0\}$ and $\Theta^1 = \{C^1, A^1\}$. Picture (d) represents the status of the Θ_L^r-tree when processing $t = 25$. Since $\text{Env}^2(\Theta^0, \Theta^1) = \text{Env}^2_{\text{root}} = 102 > 4 \cdot 25 = 100$ the overload checking triggers a failure.

Lemma 3 *The algorithm 1 runs in* $\Theta(r^2 n \log(n))$.

Proof According to lemma 2, the lines 5 and 6 of the algorithm 1 run in $\Theta(r^2 \log(n))$. Since every task is inserted exactly once in Θ^0 and once in Θ^1, overall the computational effort is $\Theta(r^2 n \log(n))$. □

4 Robust Edge-Finding

The objective of this section is to adapt the Edge-Finding algorithm introduced in [19] for the $\text{FLEXC}([S_1, \ldots, S_n], [p_1^0, \ldots, p_n^0], [d_1, \ldots, d_n], [\mathrm{c}_1, \ldots, \mathrm{c}_n], \mathcal{C}, r)$ constraint. We present a generalization of the Edge-Finding detection rules (16) and (17) for the $\text{FLEXC}([S_1, \ldots, S_n], [p_1^0, \ldots, p_n^0], [d_1, \ldots, d_n], [\mathrm{c}_1, \ldots, \mathrm{c}_n], \mathcal{C}, r)$ constraint. Prior to expound the mechanism of our algorithms, we should be mindful that in contrast to the regular scheduling problems, where the tasks are not assumed to delay, the robust framework is not symmetrical [7]. Indeed, a task can be delayed but it cannot be brought forward. Therefore, one can not simply apply the rule of filtering earliest starting times to the symmetrically negated problem in order to filter latest completion times. Therefore, we treat the filtering of lower bounds and upper bounds independently.

The structure of this section is as follows. Section 4.1 is concerned with filtering the earliest starting times. In the section 4.1.1 we establish the generic form of the robust Edge-Finding rule and accompany that with an example. In section 4.1.2 a new variant of robust earliest energy envelope for the conjunction of two disjoint subsets of tasks is defined. In section 4.1.3 we propose an extended data structure which enables us to handle the tasks when a certain number of them are delayed. Finally, in section 4.1.4 we take advantage of the introduced data structure to present the robust Edge-Finding algorithm. Moreover, we trace few steps of the algorithms for the example provided. Section 4.1 finishes with a discussion on the time complexity of the algorithm. Section 4.2 studies the filtering of latest completion times and follows the same structure as Section 4.1.

4.1 Filtering the earliest starting times

This section discusses the Edge-Finding rule, as well as the material required for detecting precedences among the tasks and adjusting earliest starting times.

4.1.1 Robust Edge-Finding rule for filtering the earliest starting times

Let $\Theta^0 \subseteq \mathcal{I}^0, \Theta^1 \subseteq \mathcal{I}^1$ and $i^b \in (\mathcal{I}^0 \cup \mathcal{I}^1) \setminus (\Theta^0 \cup \Theta^1)$ be such that $b \in \{0, 1\}$, $|\Theta^1| \leq r - b$ and $\mathcal{I}(\Theta^0) \cap \mathcal{I}(\Theta^1) = \emptyset$. Then the following states the Edge-Finding rule

$$\mathcal{C}(\max(\mathrm{lct}^0_{\mathcal{I}(\Theta^0)}, \mathrm{lct}^1_{\mathcal{I}(\Theta^1)}) - \mathrm{est}_{\mathcal{I}(\Theta^0 \cup \Theta^1 \cup \{i^b\})}) < \mathrm{e}_{\Theta^0} + \mathrm{e}_{\Theta^1} + \mathrm{e}_{i^b} \Rightarrow \Theta^0 \cup \Theta^1 \lessdot i^b \tag{29}$$

task	est	lct^0	p^0	d	c
A	1	13	4	1	6
B	6	23	9	1	7
C	16	36	8	10	2
D	14	28	12	1	6

Table 2 The set of tasks $\mathcal{I} = \{A, B, C, D\}$ to execute on a resource of capacity $\mathcal{C} = 7$ in Example 2.

Equation (29) is implied from the fact that if the execution of i^b along with a set $\Theta^0 \cup \Theta^1$ with at most $r - b$ delayed tasks causes a failure due to the overload checking, then i^b should complete after $\Theta^0 \cup \Theta^1$.

Example 2 Let $\mathcal{I} = \{A, B, C, D\}$ be the set of tasks in table 2 which must execute on a resource of capacity $\mathcal{C} = 7$ in the context where at most $r = 1$ tasks are allowed to delay. According to (29), the precedence $\{B^0, D^0\} \prec C^1$ is deduced for $\Theta^0 = \{B^0, D^0\}, \Theta^1 = \emptyset, i = C$ and $b = 1$, as 154 = 7(28 - 6) < 63 + 72 + 36 = 171 and the precedence $\{A^0, B^1\} \prec D^0$ holds for $\Theta^0 = \{A^0\}, \Theta^1 = \{B^1\}, i = D$ and $b = 0$, as 161 = 7(24 - 1) < 24 + 70 + 72 = 166.

4.1.2 Robust $\Lambda-$earliest energy envelope

The Edge-Finding can be implemented such that during the processing of the tasks, the regular or delayed tasks which make the precedences are maintained in a subset of tasks $\Lambda \subset \mathcal{I}$. Initially the tasks belong to Θ^0 and Θ^1 and the idea is to check whether adding one task from Λ^0 to Θ^0 or adding one task from Λ^1 to Θ^1 leads to $\text{Env}(\Theta^0, \Theta^1) > \mathcal{C} \cdot \max(\text{lct}^0_{\mathcal{I}(\Theta^0)}, \text{lct}^1_{\mathcal{I}(\Theta^1)})$. As soon as such a task is found in Λ, the established precedence is recorded and the task gets unscheduled from Λ. Assuming that Λ^0 and Λ^1 respectively contain the regular and delayed tasks from Λ, a variant of the earliest energy envelope of the tasks in $\Theta \cup \Lambda$, when one task from Λ is selected and at most r tasks are delayed, is defined as follows.

$$\text{Env}^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1) = \max(\max_{i^0 \in \Lambda^0} \text{Env}^r(\Theta^0 \cup \{i^0\}, \Theta^1), \max_{i^1 \in \Lambda^1} \text{Env}^r(\Theta^0, \Theta^1 \cup \{i^1\})) \tag{30}$$

$\text{Env}^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1)$ is the largest envelope that can be taken by taking a task from Λ^0 or Λ^1 and adding to Θ^0 and Θ^1. In the following, we present a data structure from which $\text{Env}^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1)$ can be retrieved efficiently.

4.1.3 $(\Theta - \Lambda)^r_L-$tree

Vilím [19] extends the $\Theta-$tree to a $(\Theta - \Lambda)-$tree. This tree is primarily different from $\Theta-$tree in that it keeps track of the tasks in Λ for which a precedence can exist. This is a feature that is not required to be addressed in the $\Theta-$tree. Similar to the $\Theta-$tree, the tasks are sorted by the earliest starting times as the leaves of $(\Theta - \Lambda)-$tree. Furthermore, the nodes of $(\Theta - \Lambda)-$tree maintain additional parameters for the energy and envelope of the tasks which belong to Λ. Analogous to the notion of Θ^r_L-tree, we define the $(\Theta - \Lambda)^r_L-$tree which is an extension of $(\Theta - \Lambda)-$tree. In addition to the parameters e^k_v and Env^k_v as in (23) and (24), this tree maintains the parameters $\Lambda-$energy, denoted $\mathrm{e}_v^{\Lambda k}$, and $\Lambda-$earliest energy envelope, denoted $\mathrm{Env}_v^{\Lambda k}$, associated to the tasks in Λ as follows.

$$\mathrm{e}_v^{\Lambda k} = \begin{cases} \mathrm{c}_v\, p_v^0 & \text{if } (v^0 \in \Lambda^0) \wedge (k = 0 \;\vee\; v^1 \notin \Lambda^1) \\ \mathrm{c}_v\, p_v^1 & \text{if } (v^1 \in \Lambda^1) \wedge (k > 0) \\ -\infty & \text{otherwise} \end{cases} \tag{31}$$

$$\mathrm{Env}_v^{\Lambda k} = \begin{cases} \mathcal{C}\, \mathrm{est}_v + \mathrm{e}_v^{\Lambda k} & \text{if } (v^0 \in \Lambda^0) \cup (v^1 \in \Lambda^1) \\ -\infty & \text{otherwise} \end{cases} \tag{32}$$

Similar to the Θ_L^r-tree, the superscript k stands for an upper bound on the number of delayed tasks in the leaf v. In order to make the computations of the $\Lambda-$energy and $\Lambda-$earliest energy envelope symmetrical over all nodes of the tree, we suppose that $0 \le k \le r$ and since $\mathrm{e}_v^{\Lambda k}$ and $\mathrm{Env}_v^{\Lambda k}$ are the $\Lambda-$energy and $\Lambda-$earliest energy envelope when at most k tasks are delayed, for $k \ge 2$ in the leaves of the $(\Theta - \Lambda)_L^r-$tree we necessarily have $\mathrm{e}_v^{\Lambda k} = \mathrm{e}_v^{\Lambda 1}$ and $\mathrm{Env}_v^{\Lambda k} = \mathrm{Env}_v^{\Lambda 1}$.

Let w be an internal node of the tree. $\mathrm{e}_w^{\Lambda k}$ and $\mathrm{Env}_w^{\Lambda k}$ are the maximum $\Lambda-$energy and the $\Lambda-$earliest energy envelope of the tasks in Θ whose leaves are descendant of w and to which one task from Λ is added. The task from Λ is also a descendant of w. It is proven [19] that

$$\mathrm{e}_w^{\Lambda 0} = \max(\mathrm{e}_{\mathrm{left}(w)}^0 + \mathrm{e}_{\mathrm{right}(w)}^{\Lambda 0}, \mathrm{e}_{\mathrm{left}(w)}^{\Lambda 0} + \mathrm{e}_{\mathrm{right}(w)}^0) \tag{33}$$

$$\mathrm{Env}_w^{\Lambda 0} = \max(\mathrm{Env}_{\mathrm{left}(w)}^{\Lambda 0} + \mathrm{e}_{\mathrm{right}(w)}^0, \mathrm{Env}_{\mathrm{right}(w)}^{\Lambda 0}, \mathrm{Env}_{\mathrm{left}(w)}^0 + \mathrm{e}_{\mathrm{right}(w)}^{\Lambda(0)}) \tag{34}$$

When scheduling, the regular tasks i^0 are added to Λ^0 and the delayed tasks i^1 are added to Λ^1. When unscheduling, the regular tasks i^0 are removed from Λ^0 and the delayed tasks i^1 are removed from Λ^1. Then, the nodes corresponding to the task are updated according to (31) and (32).

At most k tasks are delayed and in the computation of $\mathrm{e}_w^{\Lambda k}$ one task from Λ contributes. This task could be among at most j delayed tasks in the left subtree or at most $k - j$ tasks in the right subtree emanating from the inner node w, $0 \le k \le r, 0 \le j \le k$. According to (33) and (34)

$$\mathrm{e}_w^{\Lambda k} = \max_{0 \le j \le k} \{\mathrm{e}_{\mathrm{left}(w)}^j + \mathrm{e}_{\mathrm{right}(w)}^{\Lambda(k-j)}, \mathrm{e}_{\mathrm{left}(w)}^{\Lambda j} + \mathrm{e}_{\mathrm{right}(w)}^{k-j}\} \tag{35}$$

$$\mathrm{Env}_w^{\Lambda k} = \max_{0 \le j \le k} \{\mathrm{Env}_{\mathrm{left}(w)}^{\Lambda j} + \mathrm{e}_{\mathrm{right}(w)}^{k-j}, \mathrm{Env}_{\mathrm{left}(w)}^j + \mathrm{e}_{\mathrm{right}(w)}^{\Lambda(k-j)}\} \cup \{\mathrm{Env}_{\mathrm{right}(w)}^{\Lambda k}, \mathrm{Env}_w^{\Lambda(k-1)}\} \tag{36}$$

In the case that the number of delayed task is greater than the number of available nodes in the subtree of the right side, it is sufficient to retrieve $\mathrm{Env}_w^{\Lambda(k-1)}$ in (36), i.e. the lambda energy envelope when at most $k - 1$ tasks are delayed. Finally, if w is the root of $(\Theta - \Lambda)_L^r-$tree, the function $\mathrm{Env}^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1)$ can be computed by computing the value $\mathrm{Env}_{\mathrm{root}}^{\Lambda k}$ at the root node.

4.1.4 Robust Edge-Finding algorithm for filtering the earliest starting times

The implementation of Edge-Finding proceeds in two phases. Firstly, the existing precedences among the tasks are detected. Thereafter, the earliest starting time of the tasks subject to a precedence are adjusted.

Detection phase

Algorithm 2 adapts the detection phase of the Edge-Finding for the earliest starting times. This algorithm emulates the algorithm that Vilím [19] proposes. The algorithm starts with a full $(\Theta - \Lambda)^r_L$−tree, in which all the regular as well as the delayed tasks are scheduled in $\Theta = \Theta^0 \cup \Theta^1$ and $\Lambda = \Lambda^0 \cup \Lambda^1$ is empty. That is,

$$\Theta^0 = \mathcal{I}^0, \Theta^1 = \mathcal{I}^1, \Lambda^0 = \Lambda^1 = \emptyset$$

The algorithm iterates over the set of all latest completion times and delayed latest completion times $T = \{\mathrm{lct}^0_i : i \in \mathcal{I}\} \cup \{\mathrm{lct}^1_i : i \in \mathcal{I}\}$ in non-increasing order. First, the algorithm makes sure that the overload checking does not fail (line 10). If so, for every $t \in T$

$$\mathrm{Env}^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1) > \mathcal{C} \cdot t \tag{37}$$

is checked which captures the precedence. Thanks to the structure of the $(\Theta - \Lambda)^r_L$−tree, $\mathrm{Env}^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1)$ is retrieved from the root of $(\Theta - \Lambda)^r_L$−tree for each t. Line 13 retrieves the task subject to a precedence. It can be implemented in $O(\log(n))$

Algorithm 2: DetectionPhaseOfEdge-FindingForLowerBounds($\mathcal{I}$, $\mathcal{C}$, r)

```
1  $T \leftarrow \{\mathrm{lct}^0_l : l \in \mathcal{I}\} \cup \{\mathrm{lct}^1_l : l \in \mathcal{I}\}$;
2  for $i \in \{1, ..., n\}$ do
3      prec[$i$, 0] $\leftarrow -\infty$;
4      prec[$i$, 1] $\leftarrow -\infty$;
5  $\Theta^0 \leftarrow \mathcal{I}^0$;
6  $\Theta^1 \leftarrow \mathcal{I}^1$;
7  $\Lambda^0 \leftarrow \emptyset$;
8  $\Lambda^1 \leftarrow \emptyset$;
9  for $t \in T$ in non-increasing order do
10     if $\mathrm{Env}^r(\Theta^0, \Theta^1) > \mathcal{C} \cdot t$ then
11         fail;
12     while $\mathrm{Env}^{\Lambda r}_{\mathrm{root}} > \mathcal{C} \cdot t$ do
13         $i^b \leftarrow$ The task in $\Lambda^0 \cup \Lambda^1$ that maximizes $\mathrm{Env}^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1)$ ;
14         prec[$i, b$] $\leftarrow t$ ;
15         $\Lambda^b \leftarrow \Lambda^b \setminus \{i^b\}$;
16     $\Delta^1 = \{i^1 \in \mathcal{I}^1 \mid \mathrm{lct}^1_i = t\}$;
17     $\Delta^0 = \{i^0 \in \mathcal{I}^0 \mid \mathrm{lct}^0_i = t\}$;
18     $\Lambda^1 \leftarrow \Lambda^1 \cup \Delta^1$;
19     $\Theta^1 \leftarrow \Theta^1 \setminus \Delta^1$;
20     $\Lambda^0 \leftarrow \Lambda^0 \cup \Delta^0$;
21     $\Theta^0 \leftarrow \Theta^0 \setminus \Delta^0$;
```

time by traversing down the tree. The algorithm proceeds by traversing down the $(\Theta - \Lambda)^r_L-$tree from the root. At each inner node w in such a traversal, the algorithm determines which one of the cases in (36) satisfy for $\text{Env}^{\Lambda k}_w$. So long as this task is taken from $\text{Env}^{\Lambda k}_f$, where f is a child of w, the algorithm continues down. As soon as the task is taken from an $\text{e}^{\Lambda k}_f$, the algorithm switches to check the cases of (35). The task subject to a precedence is a task $i^b \in \Lambda^b, b \in \{0,1\}$, and only one task in $\Lambda^0 \cup \Lambda^1$ is used to compute $\text{Env}^{\Lambda k}_w$. The candidate task with such a property is called the *responsible task*. The responsible task, which is located on a leaf of the $(\Theta - \Lambda)^r_L-$tree, causes (30) to be maximized, hence making a precedence. Contingent upon the delay status of i, we encode the precedence that i creates in a two dimensional matrix by 0 and 1 columns, indicating whether the task is regular or delayed (line 14). The precedences detected during the detection phase are encoded in the *prec* matrix. For $b \in \{0,1\}$, $\text{prec}[i,b] = t$ means that i^b is preceded by the subsets of tasks from $\Theta^0 \cup \Theta^1$ with at most $r - b$ delayed tasks which terminate no later than t and cause (30) to be maximized. Once the precedence is recorded one line 14, the algorithm unschedules i^b from Λ^b. After the execution of the loop at line 12, all the tasks whose delayed latest completion time equals t are unscheduled from Θ^1 and they are rather scheduled in Λ^1. Furthermore, all the tasks whose latest completion time equals t, are removed from Θ^0 and Θ^1 and scheduled in Λ^0 and Λ^1.

In order to elucidate the mechanism of the algorithm 2, in figure 2 we present the first few steps of the detection phase for the tasks of example 2. For this example, $T = \{46, 36, 29, 28, 24, 23, 14, 13\}$. Figure 2a illustrates the initialization step of the algorithm, where the $(\Theta - \Lambda)^r_L-$tree is full. That is, all the regular as well as delayed tasks are scheduled and no task is a candidate yet to create a precedence.

In the first iteration $t = 46$ is processed, which corresponds to $\text{lct}^1_C = t$. C^1 gets unscheduled from Θ^1 and rather scheduled in Λ^1, to be flagged as a delayed tasks which could create a precedence later. The updated status of the node corresponding to C^1 is depicted in figure 2b. In the second iteration, $t = 36$ is processed, which corresponds to $\text{lct}^0_C = t$. At this point, $\text{Env}^{\Lambda r}_{\text{root}} = 213 \not> \mathcal{C} \cdot t$, which implies that (30) is not great enough to detect a precedence. Therefore, the loop dedicated to detect an existing precedence fails to execute and C^0 is removed from Θ^0 and fully scheduled in Λ. Figure 2c corresponds to this case after updating the entire tree. The next iteration processes $t = \text{lct}^1_D = 29$. Since $\text{Env}^{\Lambda r}_{\text{root}}$ =213 > $\mathcal{C} \cdot t$, the while loop executes. A traversal down the tree to find the responsible task locates C^1. Once the precedence $\text{prec}[C^1, 1] = 29$ is recorded, C^1 gets unscheduled from Λ^1, as illustrated in figure 2d. For this iteration there is no more precedences to detect. Thus, the while loop terminates and D^1 for which $\text{lct}^1_D = 29$ gets unscheduled from Θ^1 and scheduled in Λ^1 as depicted in figure 2e. In further executions of the algorithm, the precedence $\{A^0, B^1\} \prec D^0$ will be detected, as well. Table 3 indicates the *prec* matrix as the result of the algorithm 2.

Adjustment phase

The adjustment of the earliest starting times is done by iterating over the detected precedences which are recorded in the prec matrix. Let i^b be the tasks for which a precedence was detected with (29) and and $\text{prec}[i,b] = t$. For $\Omega^0 \subseteq \Theta^0, \Omega^1 \subseteq$

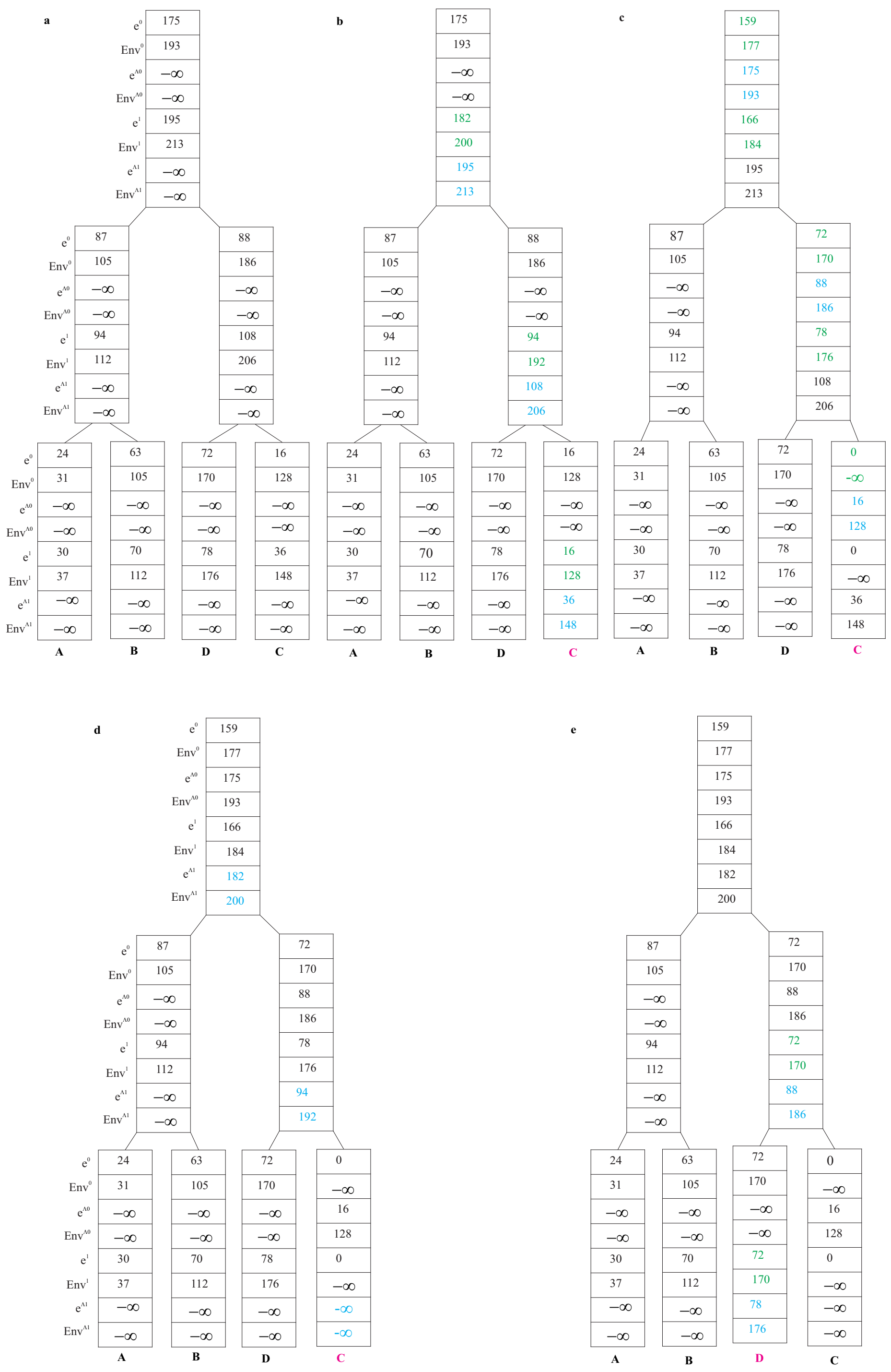


Fig. 2 In the picture (I), the algorithm starts with a full $(\Theta - \Lambda)^r_L-$tree. In the picture 2b, C^1 gets unscheduled from Θ^1 and scheduled in Λ^1. The modifications to Θ and Λ sets are respectively coloured in green and blue. In the picture 2c, C^0 is unscheduled from Θ^0 and scheduled in Λ^0. In the picture 2d after C^1 is found as the responsible tasks, it gets unscheduled from Λ^1. In the picture 2e, D^1 gets unscheduled from Θ^1 and scheduled in Λ^1.

Task \ b	0	1
A	$-\infty$	$-\infty$
B	14	14
C	28	29
D	24	24

Table 3 The *prec* matrix which is obtained after the execution of the algorithm 2 on the instance of example 2.

$\Theta^1, \emptyset \neq \Omega^0 \cup \Omega^1$ such that $|\Omega^1| \leq r - b$, we define a variation of (18) in the robust context as:

$$\mathrm{rest}(\Omega^0, \Omega^1, \mathrm{c}_i) = \mathrm{e}_{\Omega^0 \cup \Omega^1} - (\mathcal{C} - \mathrm{c}_i)(\max(\mathrm{lct}^0_{\mathcal{I}(\Omega^0)}, \mathrm{lct}^1_{\mathcal{I}(\Omega^1)}) - \mathrm{est}_{\Omega^0 \cup \Omega^1}) \quad (38)$$

The following formula adjusts the earliest starting time of i.

$$\mathrm{est}_i \leftarrow \max(\mathrm{est}_i, \max_{\substack{\Omega^0 \subseteq \mathrm{Lcut}^0(t) \setminus \{i^0\} \cap \{j^0 \in \mathcal{I}^0 : \mathrm{lct}^0_j \leq t < \mathrm{lct}^1_j\} \\ \Omega^1 \subseteq \mathrm{Lcut}^1(t) \setminus \{i^1\} \\ \emptyset \neq \Omega^0 \cup \Omega^1 \\ |\Omega^1| \leq r - b \\ \mathrm{rest}(\Omega^0, \Omega^1, \mathrm{c_i}) > 0}} \{\mathrm{est}_{\Omega^0 \cup \Omega^1} + \left\lceil \frac{\mathrm{rest}(\Omega^0, \Omega^1, \mathrm{c}_i)}{\mathrm{c}_i} \right\rceil\}) \quad (39)$$

From the condition $\mathrm{rest}(\Omega^0, \Omega^1, \mathrm{c}_i) > 0$ we obtain

$$(\mathcal{C} - \mathrm{c}_i)(\mathrm{est}_{\Omega^0 \cup \Omega^1}) + \mathrm{e}_{\Omega^0 \cup \Omega^1} > (\mathcal{C} - \mathrm{c}_i)(\max(\mathrm{lct}^0_{\mathcal{I}(\Omega^0)}, \mathrm{lct}^1_{\mathcal{I}(\Omega^1)})) \quad (40)$$

The left side of (40) describes an equivalent of $\mathrm{Env}^r(\Theta^0, \Theta^1)$, defined by (22), on a resource with capacity $\mathcal{C} - \mathrm{c}_i$. This motivates the idea of defining a variant of the earliest energy envelope, defined as (4), with respect to each $c \in F$ [19] as

$$\mathrm{Env}^{\mathrm{c}}_{\Theta} = \max_{\Omega \subseteq \Theta}((\mathcal{C} - \mathrm{c})\,\mathrm{est}_{\Omega} + \mathrm{e}_{\Omega}) \quad (41)$$

and a variant of (22) as

$$\mathrm{Env}^{\mathrm{c}\,r}(\Theta^0, \Theta^1) = \max_{\substack{\Omega^0 \subseteq \Theta^0 \\ \Omega^1 \subseteq \Theta^1 \\ |\Omega^1| \leq r \\ \mathcal{I}(\Omega^0) \cap \mathcal{I}(\Omega^1) = \emptyset}}((\mathcal{C} - \mathrm{c})\,\mathrm{est}_{\mathcal{I}(\Omega^0 \cup \Omega^1)} + \mathrm{e}_{\Omega^0} + \mathrm{e}_{\Omega^1}) \quad (42)$$

The algorithm 5 is our adaption of the adjustment phase of the Edge-Finding for this purpose. For the rest of this section, we explain how this algorithm emulates the algorithm that Vilím proposes [19].

Let F be the set of all distinct capacities. In [19] the adjustment is done by processing all $\mathrm{c} \in F$ in an outer loop and initializing a Θ^{c}-tree for each $\mathrm{c} \in F$ in an inner loop. Θ^{c}-tree is an extension of Θ-tree in that in addition to maintaining the parameters energy and earliest energy envelope, every inner node holds the additional

Algorithm 3: Maxest(tree,bound,c,$\mathcal{C}$,k,o)

```
1  $v \leftarrow \text{root}$;
2  $\text{e}[0...k] \leftarrow \mathbf{0}$ ;
3  $\text{maxEnv}^c \leftarrow (\mathcal{C} - \text{c}) \cdot \text{bound}$;
4  $k \leftarrow k - o$;
5  while $v$ is not a leaf do
6      branchRight $\leftarrow$ false;
7      for $j = 0, ..., k$ do
8          if $\text{Env}^{c,j}_{\text{right(v)}} + \text{e}[k - j] > \text{maxEnv}^c$ then
9              $v \leftarrow \text{right}(v)$;
10             branchRight $\leftarrow$ true;
11             break;
12     if not branchRight then
13         $v \leftarrow \text{left(v)}$;
14         for $j = 0, ..., k$ do
15             $\text{e}'[j] \leftarrow \max_{0 \le i \le j}(\text{e}[i] + \text{e}^{j-i}_{\text{right(v)}}, \text{e}'[j-1])$
16         $\text{e} \leftarrow \text{e}'$;
17 return $v$;
```

parameter (41). Analogous to the parameters (3) and (4) in the $\Theta-$tree, for a leaf v of the $\Theta^{\text{c}}-$tree, which corresponds to a task $i \in \mathcal{I}$, (41) is computed for $\Theta = \{i\}$ and for an inner node w of the $\Theta^{\text{c}}-$tree, (41) is computed for the set of all tasks which are

Algorithm 4: EnvelopeForLowerBound(v,tree,k)

```
1  $\text{e}_\alpha \leftarrow [\text{e}^0_v, \text{e}^1_v, 0, ..., 0]$; // $\text{e}_\alpha$ includes $k$ - 1 entries 0
2  $\text{e}_\beta \leftarrow \mathbf{0}$ ;
3  $\text{Env}_\alpha \leftarrow [\text{Env}^0_v, \text{Env}^1_v, -\infty, ..., -\infty]$; // $\text{Env}_\alpha$ includes $k$ - 1 entries $-\infty$
4  while $v$ is not the root do
5      if $v$ is a left child then
6          $\text{e}'_\beta \leftarrow \mathbf{0}$ ;
7          for $j = 0$ to $k$ do
8              $\text{e}'_\beta[j] \leftarrow \max_{0 \le i \le j}(\text{e}_\beta[i] + \text{e}^{j-i}_{\text{sibling}(v)}, \text{e}'_\beta[j-1])$
9          $\text{e}_\beta \leftarrow \text{e}'_\beta$;
10     else
11         $\text{Env}'_\alpha \leftarrow [-\infty, \ldots, -\infty]$;
12         $\text{e}'_\alpha \leftarrow \mathbf{0}$ ;
13         for $j = 0, ..., k$ do
14             $\text{Env}'_\alpha[j] \leftarrow$
                 $\max(\max_{0 \le i \le j}(\text{Env}^i_{\text{sibling}(v)} + \text{e}_\alpha[j-i]), \text{Env}_\alpha[j], \text{Env}'_\alpha[j-1])$;
15             $\text{e}'_\alpha[j] \leftarrow \max(\max_{0 \le i \le j}(\text{e}^i_{\text{sibling}(v)} + \text{e}_\alpha[j-i]), \text{e}'_\alpha[j-1])$;
16     $\text{Env}_\alpha \leftarrow \text{Env}'_\alpha$;
17     $\text{e}_\alpha \leftarrow \text{e}'_\alpha$;
18     $v \leftarrow \text{parent}(v)$;
19 return $\max_{0 \le i \le k}(\text{e}_\beta[i] + \text{Env}_\alpha[k-i])$;
```

Algorithm 5: AdjustmentOfLowerBounds(prec, r, $\mathcal{C}$)

```
$F \leftarrow$ The capacities of the tasks for which a precedence was detected;
for $\mathrm{c} \in F$ do
    $\Theta^0 \leftarrow \emptyset$;
    $\Theta^1 \leftarrow \emptyset$;
    upd $\leftarrow [-\infty, -\infty]$;
    $t' \leftarrow 0$;
    for $i^b \in \{j^b \mid prec[j,b] > -\infty \wedge \mathrm{c}_i = \mathrm{c}\}$ in non-deccreasing order of prec[i, b] do
        $t \leftarrow \mathrm{prec}[i,b]$;
        $\Theta^1 \leftarrow \Theta^1 \cup (\mathrm{Lcut}^1(t) \setminus \mathrm{Lcut}^1(t')) \setminus \{i^1\}$ ;
        $\Theta^0 \leftarrow \Theta^0 \cup \{j^0 \in \mathcal{I}^0 : \mathrm{lct}^0_j \leq t < \mathrm{lct}^1_j\} \setminus \{j^0 \in \mathcal{I}^0 : \mathrm{lct}^0_j \leq t' < \mathrm{lct}^1_j\}) \setminus \{i^0\}$ ;
        $m \leftarrow \max(\{\mathrm{lct}^0_j \mid j^0 \in \Theta^0\} \cup \{\mathrm{lct}^1_j \mid j^1 \in \Theta^1\})$ ;
        $v \leftarrow \mathrm{Maxest}(\Theta, m, c, \mathcal{C}, r, b)$;
        Env $\leftarrow \mathrm{EnvelopeForLowerBound}(v, \Theta, r - b)$ ;
        diff $\leftarrow \lceil(\mathrm{Env} - (\mathcal{C} - \mathrm{c})m/\,\mathrm{c}\rceil$ ;
        upd[b] $\leftarrow$ max(upd[b],diff) ;
        $\mathrm{est}_i \leftarrow \max(\mathrm{est}_i$, upd[b] );
        if $\mathrm{lct}^0_i \leq t \wedge \mathrm{lct}^1_i \leq t$ then $\Theta^1 \leftarrow \Theta^1 \cup \{i^1\}$;
        else if $\mathrm{lct}^0_i \leq t$ then $\Theta^0 \leftarrow \Theta^0 \cup \{i^0\}$ ;
        $t' \leftarrow t$;
```

included in the subtree rooted at w. This computation is done recursively as below.

$$\mathrm{Env}^{\mathrm{c}}_w = \max(\mathrm{Env}^{\mathrm{c}}_{\mathrm{left}(w)} + \mathrm{e}^0_{\mathrm{right}(w)}, \mathrm{Env}^{\mathrm{c}}_{\mathrm{right}(w)}) \tag{43}$$

Note that in [19] all distinct capacities $\mathrm{c} \in F$ are considered individually, no matter a precedence was detected per capacity or not. For the sake of efficiency, we rather consider F as the set of capacities of the tasks for which a precedence was detected. For each $\mathrm{c} \in F$, we initialize an empty $\Theta^{\mathrm{c}\,r}_L-$tree, which is an extension of Θ^r_L-tree. With regard to the material presented in section 3.4, a leaf v of the $\Theta^{\mathrm{c}\,r}_L-$tree in addition to (23) and (24) holds

$$\mathrm{Env}^{\mathrm{c}\,k}_v = \begin{cases} (\mathcal{C} - \mathrm{c})\,\mathrm{est}_v + \mathrm{e}^k_v & \text{if } (v^0 \in \Theta^0) \vee (v^1 \in \Theta^1) \\ -\infty & \text{otherwise} \end{cases} \tag{44}$$

and $\mathrm{Env}^{\mathrm{c}\,k}_w$ for an internal node w is computed as

$$\mathrm{Env}^{\mathrm{c}\,k}_w = \max_{0 \leq j \leq k}\{\mathrm{Env}^{\mathrm{c}\,j}_{\mathrm{left}(w)} + \mathrm{e}^{k-j}_{\mathrm{right}(w)}\} \cup \{\mathrm{Env}^{\mathrm{c}\,k}_{\mathrm{right}(w)}, \mathrm{Env}^{\mathrm{c}(k-1)}_w)\} \tag{45}$$

Θ^{c}-tree develops by processing all lct^0_j for $j \in \mathcal{I}$ in non-decreasing order and scheduling j by adding it to Θ. Rather than iterating through all lct^0_j and lct^1_j for $j \in \mathcal{I}$, we develop the $\Theta^{\mathrm{c}\,r}_L-$tree by iterating over the tasks i^b in non-decreasing order of $\mathrm{prec}[i,b]$. The adjustment should be done only if $\mathrm{c}_i = \mathrm{c}$. If so, at iterating each $\mathrm{prec}[i,b] = t$, all the regular or delayed tasks which cannot complete after t are scheduled in the $\Theta^{\mathrm{c}\,r}_L-$tree by adding to Θ^0 or Θ^1.

Let $\Omega^0 \cup \Omega^1$ be the candidate subset which can give the strongest adjustment in (39). Once all the tasks which are eligible to be in $\Omega^0 \cup \Omega^1$ are scheduled in the tree, it is time to identify Ω^0 and Ω^1 and compute the value of the second component of the right side of (39). Emulating [19], we proceed to compute Ω^0 and Ω^1 in two steps.

The first step is to compute the maximum earliest starting time (or *maxest*) for $\Omega^0 \cup \Omega^1$. There might be multiple sets $\Omega^{0\prime}$ and $\Omega^{1\prime}$ that satisfy (39) and the goal is to compute the largest $\mathrm{est}_{\Omega^{0\prime} \cup \Omega^{1\prime}}$ for the appropriate $\Omega^{0\prime}$ and $\Omega^{1\prime}$. The new variant of the earliest energy envelope, defined in (41), helps to locate the node responsible for maxest. In the algorithms 3, we consider this idea for identifying maxest for $\Omega^0 \cup \Omega^1$ in (39). Note that line 11 of algorithm 5 retrieves the largest completion time for the set of all tasks that are scheduled so far, whether they are regular or delayed. This value, named *bound*, which is given as a parameter to the algorithm 3, is important for checking the right side of (40) in this algorithm.

The second step is to compute the envelope of the tasks which is done in the algorithm 4 . This is done by dividing the tasks in two groups. The tasks which start before or at maxest and the rest of the tasks which start after maxest. The tasks which qualify for the former case belong to a set $\alpha(j, \mathrm{c})$ and the tasks which qualify for the latter case belong to a set $\beta(j, c)$. From the sets $\alpha(j, \mathrm{c})$ and $\beta(j, c)$, the earliest energy envelop in the $\Theta_L^{\mathrm{c}\,r}-$tree is computed with

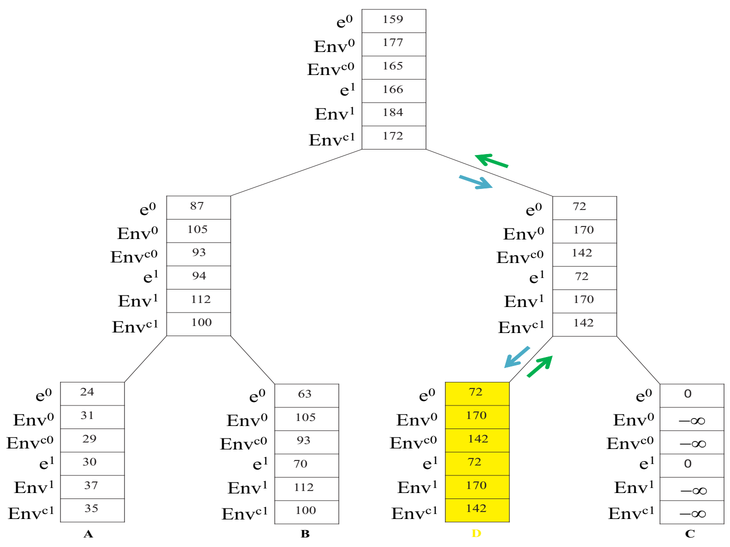
$$\mathrm{Env}(j, c) = \mathrm{e}_\beta + \mathrm{Env}_{\alpha(j,\mathrm{c})} \tag{46}$$

where e_β and Env_α are computed in a bottom-up manner, starting from the located task responsible for maxest, and by taking into account that at most r tasks are delayed. Once the envelope is computed, the value which makes the strongest update is adjusted by taking the maximum between the envelope computed in the current iteration and the preceding iterations (line 15). Ultimately, if the task can not finish after t, it gets scheduled itself.

Here is the adjustment of the precedence $\{B^0, D^0\} \prec C^1$ for the task C from the example 2. Figure 3 illustrates the $\Theta_L^{\mathrm{c}\,r}-$tree for the state where est_C is adjusted. The adjustment is done when processing $\mathrm{c} = 2$. At this state, $\Theta^0 = \{A^0, B^0, D^0\}$ and $\Theta^1 = \{A^1, B^1\}$. The set of scheduled tasks has, for largest completion completion time, $m = 28$ in line 11 of the algorithm. The blue arrows in figure 3 show the path which is traversed by the algorithm 3. This algorithm locates the yellow node which corresponds to the task D for maxest. The algorithm 4 starts from the yellow node and the green arrows show the path which is actually the reversed blue path traversed by this algorithm to the root. This algorithm computes $\mathrm{Env} = 184$ for line 13 and in line 14 the difference is computed diff = $\lceil 184 - (7-2)28/2 \rceil = 22$. Hence, est_C gets filtered to $\mathrm{est}_C = 22$. Proceeding the adjustment phase for the precedence $\{A^0, B^1\} \prec D^0$ in a similar fashion yields $\mathrm{est}_D = 15$.

Lemma 4 *The time complexity of the robust Edge-Finding is $O(r^2 kn \log(n))$.*

Proof Unscheduling a task from Θ^0 or Θ^1 or scheduling in Λ^0 or Λ^1 requires to update the values e_i and Env_i, $\mathrm{e}_v^{\Lambda j}$ and $\mathrm{Env}_v^{\Lambda j}$ of the leaf of the tree as well as all nodes up to the root of the tree. Therefore, the lines 18, 19, 20 and 21 of the algorithm 2 run in $O(r^2 \log(n))$. This complexity is maintained for finding the responsible task, finding the maxest as well as computing the envelope, since for such operations the tree is traversed from the root to the leaves or conversely. The scheduling and unscheduling



e0 159 Env0 177 Envc0 165 e1 166 Env1 184 Envc1 172
e0 87 Env0 105 Envc0 93 e1 94 Env1 112 Envc1 100
e0 72 Env0 170 Envc0 142 e1 72 Env1 170 Envc1 142
e0 24 Env0 31 Envc0 29 e1 30 Env1 37 Envc1 35
A
e0 63 Env0 105 Envc0 93 e1 70 Env1 112 Envc1 100
B
e0 72 Env0 170 Envc0 142 e1 72 Env1 170 Envc1 142
D
e0 0 Env0 −∞ Envc0 −∞ e1 0 Env1 −∞ Envc1 −∞
C

Fig. 3 The state of $\Theta_L^{c\,r}-$tree in the iteration that est_C gets adjusted. The blue arrows show the path which is traversed by the algorithm 3. This algorithm locates the yellow node. The algorithm 4 starts from the yellow node and the green arrows show the path traversed by this algorithm to the root.

tasks at lines 9 and 10 of the algorithm 5 occur at most n times, each time implying in $O(r^2 \log(n))$ computations. Moreover, since each task is unscheduled once from Θ^0 and once from Θ^1 or scheduled once in Λ^0 and once in Λ^1, considering that there are k distinct capacities, the overall time complexity is $O(r^2 kn \log(n))$. □

4.2 Filtering the latest completion times

This section discusses the Edge-Finding rule, as well as the material required for detecting precedences among the tasks and adjusting latest completion times.

4.2.1 Robust Edge-Finding rules for filtering the latest completion times

Let $\Theta^0 \subseteq \mathcal{I}^0, \Theta^1 \subseteq \mathcal{I}^1$ and $i^0 \in \mathcal{I}^0 \setminus \Theta^0$ be such that $\mathcal{I}(\Theta^0) \cap \mathcal{I}(\Theta^1) = \emptyset$. The following states the first Edge-Finding rule

$$\mathcal{C}(\max(\mathrm{lct}^0_{\mathcal{I}(\Theta^0)\cup\{i\}}, \mathrm{lct}^1_{\mathcal{I}(\Theta^1)}) - \mathrm{est}_{\mathcal{I}(\Theta^0\cup\Theta^1)}) < \mathrm{e}_{\Theta^0} + \mathrm{e}_{\Theta^1} + \mathrm{e}_{i^0} \Rightarrow i^0 \prec \Theta^0 \cup \Theta^1 \tag{47}$$

If $i^1 \in \mathcal{I}^1 \setminus \Theta^1$ be such that $\mathcal{I}(\Theta^0) \cap \mathcal{I}(\Theta^1) = \emptyset$, then the following states the second Edge-Finding rule

$$\mathcal{C}(\max(\mathrm{lct}^0_{\mathcal{I}(\Theta^0)}, \mathrm{lct}^1_{\mathcal{I}(\Theta^1)\cup\{i\}}) - \mathrm{est}_{\mathcal{I}(\Theta^0\cup\Theta^1)}) < \mathrm{e}_{\Theta^0} + \mathrm{e}_{\Theta^1} + \mathrm{e}_{i^1} \Rightarrow i^1 \prec \Theta^0 \cup \Theta^1 \tag{48}$$

task	est	lct^0	p^0	d	c
A	1	13	4	1	6
B	6	23	9	1	7
C	22	36	8	10	2
D	15	28	12	1	6

Table 4 Tasks of Example 2 with updated earliest starting times

Example 3 After adjusting est_C and est_D due to the precedences detected for the tasks C and D in example 2, the tasks are updated as in Table 4. This update causes two new precedences to be detected by filtering the latest completion times. Assuming $\Theta^0 = \{B^0\}, \Theta^1 = \{D^1\}, i = A, b = 0$, the precedence $A^0 \prec \{D^1, B^0\}$ holds as 161 = 7(29 - 6) < 24 + 63 + 78 = 165 and assuming $\Theta^0 = \{B^0\}, i = D, b = 0$ the precedence $B^0 \prec D^0$ holds as 91 = 7(28 - 15) < 63 +72 = 135.

4.2.2 Robust latest energy envelope

One can generalize the notion of the latest energy envelope, defined with (5), for the case that the tasks can be delayed. Rather than computing the latest energy envelope of a set Θ as in (5), we compute the latest energy envelope of two sets $\Theta^0 \subseteq \mathcal{I}^0$ and $\Theta^1 \subseteq \mathcal{I}^1$. The tasks in Θ^0 can be regular while the tasks in Θ^1 can be delayed. The tasks that belong to both sets can either be regular or delayed but not both. The latest energy envelope for the sets Θ^0 and Θ^1 in such a case is defined

$$\text{Env}'^r(\Theta^0, \Theta^1) = \min_{\substack{\Omega^0 \subseteq \Theta^0 \\ \Omega^1 \subseteq \Theta^1 \\ |\Omega^1| \le r \\ \mathcal{I}(\Omega^0) \cap \mathcal{I}(\Omega^1) = \emptyset}} (\mathcal{C} \max(\text{lct}^0_{\mathcal{I}(\Omega^0)}, \text{lct}^1_{\mathcal{I}(\Omega^1)}) - \text{e}_{\Omega^0} - \text{e}_{\Omega^1}) \tag{49}$$

4.2.3 Robust $\Lambda-$latest energy envelope

Similar to filtering the earliest starting times, the Edge-Finding for filtering the latest completion times can also be implemented such that during the processing of the tasks, the regular or delayed tasks which make the precedences are maintained in a subset of tasks $\Lambda = \Lambda^0 \cup \Lambda^1 \subset \mathcal{I}$. Initially the tasks belong to Θ and the idea is to check whether adding one task from Λ^0 to Θ or adding one task from Λ^1 to Θ leads to $\text{Env}'(\Theta^0, \Theta^1) < \mathcal{C} \cdot \text{est}_{\mathcal{I}(\Theta^0 \cup \Theta^1)}$ for $\Theta^0 = \{i^0 \mid i \in \Theta\}$ and $\Theta^1 = \{i^1 \mid i \in \Theta\}$. As soon as such a task is found in Λ, the established precedence is recorded and the task gets unscheduled from Λ. Assuming that Λ^0 and Λ^1 respectively contain the regular and delayed tasks from Λ, a variant of the latest energy envelope of the tasks in $\Theta \cup \Lambda$, when one task from Λ is selected and at most r tasks are delayed, is defined as follows.

$$\text{Env}'^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1) = \min(\min_{i^0 \in \Lambda^0} \text{Env}'^r(\Theta^0 \cup \{i^0\}, \Theta^1), \min_{i^1 \in \Lambda^1} \text{Env}'^r(\Theta^0, \Theta^1 \cup \{i^1\})) \tag{50}$$

$\mathrm{Env}'^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1)$ is the smallest envelope that can be taken by taking a responsible task from Λ^0 or Λ^1 and adding to Θ^0 and Θ^1. In the following, we present a data structure from which $\mathrm{Env}'^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1)$ can be retrieved efficiently.

4.2.4 $(\Theta - \Lambda)^r_U$*−tree*

Analogous to the notion of $(\Theta - \Lambda)^r_L$−tree, we define the $(\Theta - \Lambda)^r_U$−tree in order to develop the detection algorithm for filtering latest completion times. However, compared with the the preceding section, there is a major discrepancy in the way we construct the $(\Theta - \Lambda)^r_U$−tree. We associate each task i to two leaves. The regular task i^0 is associated to a leaf as usual but the delayed task i^1 is associated to another leaf that takes into account only the delayed part of the task. One can interpret it as a task with processing time d_i. Moreover, the values computed in each node of the tree are a function of three sets: $\Theta \subseteq \mathcal{I}$, $\Lambda^0 \subset \mathcal{I}^0$, and $\Lambda^1 \subset \mathcal{I}^1$. For instance, with such an interpretation, in the example 3, Λ^0 has $p^0_A \cdot \mathrm{c}_A = 4 \cdot 6 = 24$ units of energy, while i^1 has $d_A \cdot \mathrm{c}_A = 1 \cdot 6 = 6$ units of energy.

Let $T = \{\mathrm{lct}^0_i : i \in \mathcal{I}\} \cup \{\mathrm{lct}^1_i : i \in \mathcal{I}\}$ be the set of all latest completion times and delayed latest completion times sorted in ascending order. We construct the $(\Theta - \Lambda)^r_U$−tree with $2n$ leaves. The tree is initialized with $2n$ tasks: n regular tasks and n delayed tasks. The leaves are sorted by lct^0 for the regular tasks and lct^1 for the delayed tasks. For an arbitrary leaf v, the energy and envelopes are defined as

$$\mathrm{e}^k_{v^0} = \begin{cases} \mathrm{c}_v\, p^0_v & \text{if } k \geq 0 \\ 0 & \text{if } v^0 \notin \Theta \end{cases}$$

$$\mathrm{e}^k_{v^1} = \begin{cases} 0 & \text{if } k = 0 \\ \mathrm{c}_v\, d_v & \text{if } k > 0 \\ 0 & \text{if } v^1 \notin \Theta \end{cases}$$

$$\mathrm{Env}'^k_{v^0} = \begin{cases} \mathcal{C}\,\mathrm{lct}^0_v - \mathrm{e}^k_{v^0} & \text{if } k \geq 0 \\ \infty & \text{if } v^0 \notin \Theta \end{cases}$$

$$\mathrm{Env}'^k_{v^1} = \begin{cases} \infty & \text{if } k = 0 \\ \mathcal{C}\,\mathrm{lct}^1_v - \mathrm{c}_v\, d_v & \text{if } k > 0 \\ \infty & \text{if } v^1 \notin \Theta \end{cases}$$

$$\mathrm{e}^{\Lambda k}_{v^0} = \begin{cases} \mathrm{c}_v\, p^0_v & \text{if } (k \geq 0) \wedge (v^0 \in \Lambda^0) \\ -\infty & \text{if } v^0 \notin \Lambda^0 \end{cases}$$

$$\mathrm{e}^{\Lambda k}_{v^1} = \begin{cases} 0 & \text{if } (k = 0) \wedge (v^1 \in \Lambda^1) \\ (p^0_v + d_v)\,\mathrm{c}_v & \text{if } (k > 0) \wedge (v^1 \in \Lambda^1) \\ -\infty & \text{if } v^1 \notin \Lambda^1 \end{cases}$$

$$\mathrm{Env}'^{\Lambda k}_{v^0} = \begin{cases} \mathcal{C}\,\mathrm{lct}^0_v - \mathrm{c}_v\, p^0_v & \text{if } (k \geq 0) \wedge (v^0 \in \Lambda^0) \\ \infty & \text{if } v^0 \notin \Lambda^0 \end{cases}$$

$$\mathrm{Env}'^{\Lambda k}_{v^1} = \begin{cases} \infty & \text{if } (k = 0) \wedge (v^1 \in \Lambda^1) \\ \mathcal{C}\,\mathrm{lct}^1_v - (p^1_v + d_v)\,\mathrm{c}_v & \text{if } (k > 0) \wedge (v^1 \in \Lambda^1) \\ \infty & \text{if } v^1 \notin \Lambda^1 \end{cases}$$

Similar to the $(\Theta - \Lambda)^r_L-$tree, the superscript k stands for an upper bound on the number of delayed tasks in the leaf v. In order to make the computations of the $\Lambda-$energy and $\Lambda-$latest energy envelope symmetrical over all nodes of the tree, we suppose that $0 \le k \le r$ and since $\mathrm{e}^{\Lambda k}_v$ and $\mathrm{Env}'^{\Lambda k}_v$ are the $\Lambda-$energy and $\Lambda-$latest energy envelope when at most k tasks are delayed, for $k \ge 2$ in the leaves of the $(\Theta - \Lambda)^r_L-$tree we necessarily have $\mathrm{e}^{\Lambda k}_v = \mathrm{e}^{\Lambda 1}_v$ and $\mathrm{Env}'^{\Lambda k}_v = \mathrm{Env}'^{\Lambda 1}_v$.

Scheduling a regular or delayed task in $\Lambda^0 \cup \Lambda^1$ is equivalent to adding the task i^0 to Λ^0 or the task i^1 to Λ^1 and updating the node corresponding to the task according to the formulae above. Unscheduling a regular or delayed task from $\Lambda^0 \cup \Lambda^1$ is equivalent to removing the task i^0 from Λ^0 or the task i^1 from Λ^1 and updating the node corresponding to the task according to the formulae above. Unscheduling a task i from Θ removes both i^0 and i^1.

Let w be an internal node of the tree. Scott [18] proves that

$$\mathrm{Env}'^0_w = \min(\mathrm{Env}^0_{\mathrm{right}(w)} - \mathrm{e}^0_{\mathrm{left}(w)}, \mathrm{Env}'^0_{\mathrm{left}(w)}) \tag{51}$$

$\mathrm{Env}'^{\Lambda k}_w$ is the minimum $\Lambda-$latest energy envelope of the tasks in Θ whose leaves are descendant of w and to which one task from $\Lambda^0 \cup \Lambda^1$ is added. The task from $\Lambda^0 \cup \Lambda^1$ is also a descendant of w. With a reasoning similar to (28) and (36), the values of Env'^k_w and $\mathrm{Env}'^{\Lambda k}_w$ for at most k delayed tasks, $0 \le k \le r$, are recursively computed as

$$\mathrm{Env}'^k_w = \min_{0 \le j \le k} \{\mathrm{Env}'^j_{\mathrm{right}(w)} - \mathrm{e}^{k-j}_{\mathrm{left}(w)}\} \cup \{\mathrm{Env}'^k_{\mathrm{left}(w)}, \mathrm{Env}'^{(k-1)}_w\} \tag{52}$$

$$\mathrm{Env}'^{\Lambda k}_w = \min_{0 \le j \le k} \{\mathrm{Env}'^{\Lambda j}_{\mathrm{right}(w)} - \mathrm{e}^{k-j}_{\mathrm{left}(w)}, \mathrm{Env}'^j_{\mathrm{right}(w)} - \mathrm{e}^{\Lambda(k-j)}_{\mathrm{left}(w)}\} \cup \{\mathrm{Env}'^{\Lambda k}_{\mathrm{left}(w)}, \mathrm{Env}'^{\Lambda(k-1)}_w\} \tag{53}$$

In the case that the number of delayed tasks is greater than the number of available nodes in the subtree of the right side, it is sufficient to retrieve $\mathrm{Env}'^{\Lambda(k-1)}_w$ in (53), i.e. the lambda energy envelope when at most $k - 1$ tasks are delayed.

Finally, the function $\mathrm{Env}'^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1)$ for $\Theta^0 = \{i^0 \mid i \in \Theta\}$ and $\Theta^1 = \{i^1 \mid i \in \Theta\}$ can be computed by computing the value $\mathrm{Env}'^{\Lambda k}_w$ at the root node of $(\Theta - \Lambda)^r_U-$tree.

4.2.5 Robust Edge-Finding algorithm for filtering the latest completion times

Analogous to the case for filtering the earliest starting times, the implementation of Edge-Finding for filtering the latest completion times proceeds in two phases.

Algorithm 6: DetectionPhaseOfEdge-FindingForUpperBounds($\mathcal{I}$)

```
1  for $i \in \{1, ..., n\}$ do
2      prec[$i$,0] $\leftarrow \infty$;
3      prec[$i$,1]$\leftarrow \infty$;
4  $\Theta \leftarrow \mathcal{I}$;
5  $\Lambda^0 \leftarrow \emptyset$;
6  $\Lambda^1 \leftarrow \emptyset$;
7  for $j \in \mathcal{I}$ in non-decreasing order of $\mathrm{est}_j$ do
8      while $\mathrm{Env}'^{\Lambda r}_{\mathrm{root}}(\Theta, \Lambda) < \mathcal{C} \cdot \mathrm{est}_j$ do
9          $i^b \leftarrow$ The task in $\Lambda^0 \cup \Lambda^1$ that minimizes $\mathrm{Env}'^{\Lambda r}(\Theta, \Lambda)$ ;
10         prec[$i, b$] $\leftarrow \mathrm{est}_j$;
11         $\Lambda^b \leftarrow \Lambda^b \setminus \{i^b\}$;
12     $\Theta \leftarrow \Theta \setminus \{j\}$;
13     $\Lambda^0 \leftarrow \Lambda^0 \cup \{j^0\}$;
14     $\Lambda^1 \leftarrow \Lambda^1 \cup \{j^1\}$;
```

Detection phase

Algorithm 6 adapts the detection phase of the Edge-Finding for the latest completion times. The algorithm starts with a full $(\Theta - \Lambda)^r_U-$tree, in which all the regular as well as the delayed tasks are scheduled in Θ and $\Lambda = \Lambda^0 \cup \Lambda^1$ is empty. That is,

$$\Theta = \mathcal{I}^0 \cup \mathcal{I}^1, \Lambda^0 = \Lambda^1 = \emptyset$$

The algorithm iterates over the set of all earliest starting times in non-decreasing order. If the filtering of earliest starting times and latest completion times of tasks are both implemented respectively, it is not necessary to test the Overload Checking in algorithm 6. For every $j \in \mathcal{I}$ in non-decreasing order

$$\mathrm{Env}'^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1) < \mathcal{C} \cdot \mathrm{est}_j \tag{54}$$

is checked which captures the precedence. Thanks to the structure of the $(\Theta-\Lambda)^r_U-$tree, $\mathrm{Env}'^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1)$ is retrieved from the root of $(\Theta - \Lambda)^r_U-$tree for each $j \in \mathcal{I}$. Line 9 retrieves the task subject to a precedence. It can be implemented in $O(\log(n))$ time by traversing down the tree. The algorithm proceeds by traversing down the $(\Theta - \Lambda)^r_U-$tree from the root. At each inner node w in such a traversal, the algorithm determines which one of the cases in (53) satisfy for $\mathrm{Env}'^{\Lambda k}_w$. So long as this task is taken from an $\mathrm{Env}'^{\Lambda k}_f$, where f is a child of w, the algorithm continues down. As soon as the task is taken from an $\mathrm{e}^{\Lambda k}_f$, the algorithm switches to check the cases of (35). Similarly, the responsible task subject to a precedence is only one task $i^b \in \Lambda^b$ that is located on a leaf of the $(\Theta - \Lambda)^r_U-$tree and causes (50) to be minimized, hence making a precedence. We encode the precedence that i creates in a two dimensional matrix by 0 and 1 columns, named prec, which indicates whether the task is regular or delayed (line 10). For $b \in \{0, 1\}$, $prec[i, b] = \mathrm{est}_j$ for some $j \in \mathcal{I}$ means that the subsets of tasks from $\Theta^0 \cup \Theta^1$ with at most $r - b$ delayed tasks which start no earlier than j are preceded by i^b and i^b cause (50) to be minimized. Once the precedence is recorded at line 10, the algorithm unschedules i^b from Λ^b. After the execution of the

Task \ *b*	0	1
A	6	6
B	15	15
C	∞	∞
D	22	22

Table 5 The *prec* matrix which is obtained after the execution of the algorithm 6 on the instance of example 3.

loop, the tasks corresponding to j is unscheduled from Θ and rather scheduled in Λ^0 and Λ^1.

Figure 4 depicts a trace of the algorithm for the example 3 in few steps. Figure 4a illustrates the initialization step of the algorithm, where the $(\Theta - \Lambda)^r_U$−tree is full. That is, all the regular as well as delayed tasks are scheduled and no task is a candidate yet to create a precedence.

In the first iteration the task A is processed and it gets unscheduled. Thus, the two leaves of the $(\Theta - \Lambda)^r_U$−tree corresponding to A^0 and A^1 are modified. The updated status of the node corresponding to A is depicted in figure 4b. In the second iteration, B is processed. At this point, $30 < \mathcal{C} \cdot \mathrm{est}_B$, which causes the loop dedicated to detect an existing precedence to execute. Since A^1 is detected as the responsible task, it gets unscheduled. The loop executes again and A^0 is detected as responsible task. Therefore, it gets unscheduled, as illustrated in the figure 4c. In the next execution, the loop gets pasted and now B which corresponds to the time point t is unscheduled. Figure 4d represents this step. Table 5 indicates the prec matrix as the result of the algorithm 6.

Adjustment phase

For $j \in \mathcal{I}$, we define the *right cut* of j with respect to its est_j to be

$$\mathrm{Rcut}(j) = \{l \in \mathcal{I} : \mathrm{est}_j \leq \mathrm{est}_l\}$$

i.e. $\mathrm{Rcut}(j)$ includes all of the tasks which start no earlier than j.

The adjustment of the latest completion times is done by iterating over the detected precedences which are recorded in the prec matrix. Let i^b be the task for which a precedence was detected with (47) or (48) and $\mathrm{prec}[i, b] = \mathrm{est}_j$ for some $j \in \mathcal{I}$. The following formula adjusts the latest completion time of i.

$$\mathrm{lct}^0_i \leftarrow \min(\mathrm{lct}^0_i, \min_{\substack{\Omega^0 : \mathcal{I}(\Omega^0) \subseteq \mathrm{Rcut}(j) \setminus \{i\} \\ \Omega^1 : \mathcal{I}(\Omega^1) \subseteq \mathrm{Rcut}(j) \setminus \{i\} \\ \emptyset \neq \Omega^0 \cup \Omega^1 \\ |\Omega^1| \leq r - b \\ \mathrm{rest}(\Omega^0, \Omega^1, \mathrm{c_i}) > 0}} \{\max(\mathrm{lct}^0_{\mathcal{I}(\Omega^0)}, \mathrm{lct}^1_{\mathcal{I}(\Omega^1)}) - \left\lceil \frac{\mathrm{rest}(\Omega^0, \Omega^1, \mathrm{c}_i)}{\mathrm{c}_i} \right\rceil\}) \tag{55}$$

Note that in (55), Ω^0 includes the regular tasks and Ω^1 includes the delayed tasks as described in section 4.2.4.

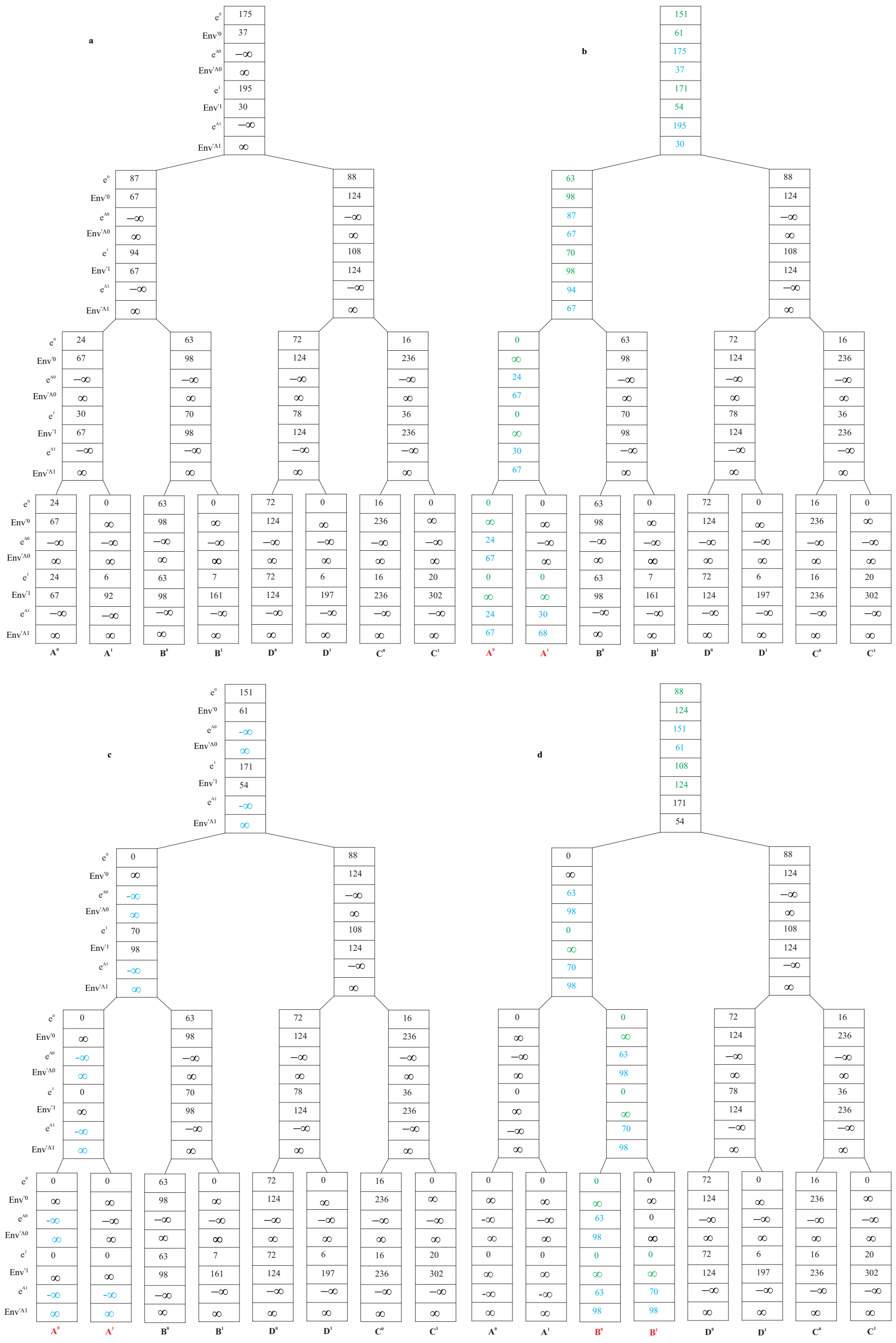


Fig. 4 In the figure 4a, the algorithm starts with a full $(\Theta-\Lambda)_U^r-$tree. In the figure 4b, A gets unscheduled from Θ and scheduled in Λ. In the figure 4c, A^1 and then A^0 are detected as responsible tasks and get unscheduled. In the figure 4d, B is unscheduled.

Algorithm 7: AdjustmentOfUpperBounds(prec,r,$\mathcal{C}$)

```
$F \leftarrow$ The capacities of the tasks for which a precedence was detected;
for c $\in F$ do
    $\Theta \leftarrow \emptyset$;
    upd $\leftarrow [\infty, \infty]$;
    $t' \leftarrow \infty$;
    for $i^b \in \{j^b \mid prec[j, b] < \infty \wedge \mathrm{c}_i = \mathrm{c}\}$ in non-increasing order of prec[i, b] do
        $t \leftarrow \mathrm{prec}[i, b]$ for some $j \in \mathcal{I}$;
        $\Theta \leftarrow \Theta \cup (\mathrm{Rcut}(j) \setminus \mathrm{Rcut}(t')) \setminus \{i\}$ ;
        $v \leftarrow \mathrm{MinIct}(\Theta, t, c, \mathcal{C}, r, b)$;
        $\mathrm{Env}' \leftarrow \mathrm{EnvelopeForUpperBound}(v, \Theta, r - b)$;
        diff $\leftarrow \lfloor (\mathrm{Env}' - (\mathcal{C} - \mathrm{c})t / \mathrm{c} \rfloor$;
        upd[b] $\leftarrow$ min(upd[b],diff);
        $\mathrm{lct}_i^0 \leftarrow \min(\mathrm{lct}_i^0, \mathrm{upd}[b])$;
        if $\mathrm{est}_i \geq t$ then
            $\Theta \leftarrow \Theta \cup \{i\}$
        $t' \leftarrow t$;
```

From the condition $\mathrm{rest}(\Omega^0, \Omega^1, \mathrm{c}_i) > 0$ we obtain

$$(\mathcal{C} - \mathrm{c}_i)(\mathrm{est}_{\Omega^0 \cup \Omega^1}) > (\mathcal{C} - \mathrm{c}_i)(\max(\mathrm{lct}^0_{\mathcal{I}(\Omega^0)}, \mathrm{lct}^1_{\mathcal{I}(\Omega^1)})) - \mathrm{e}_{\Omega^0 \cup \Omega^1} \tag{56}$$

The right side of (56) describes an equivalent of $\mathrm{Env}'^r(\Theta^0, \Theta^1)$, defined by (49), on a resource with capacity $\mathcal{C} - \mathrm{c}_i$. This motivates the idea of defining a variant of the latest energy envelope, defined as (5), with respect to each $c \in F$ as

$$\mathrm{Env}'^{\mathrm{c}}_{\Theta} = \min_{\Omega \subseteq \Theta} ((\mathcal{C} - \mathrm{c})\, \mathrm{lct}^0_{\Omega} - \mathrm{e}_{\Omega}) \tag{57}$$

and a variant of $\mathrm{Env}'^r(\Theta^0, \Theta^1)$, defined in (49), as

$$\mathrm{Env}'^{\mathrm{c}\,r}(\Theta^0, \Theta^1) = \min_{\substack{\Omega^0 \subseteq \Theta^0 \\ \Omega^1 \subseteq \Theta^1 \\ |\Omega^1| \leq r \\ \mathcal{I}(\Omega^0) \cap \mathcal{I}(\Omega^1) = \emptyset}} ((\mathcal{C} - \mathrm{c}) \max(\mathrm{lct}^0_{\mathcal{I}(\Omega^0)}, \mathrm{lct}^1_{\mathcal{I}(\Omega^1)}) - \mathrm{e}_{\Omega^0} - \mathrm{e}_{\Omega^1}) \tag{58}$$

The algorithm 7 is our adaption of the adjustment phase of the Edge-Finding for this purpose. For each $\mathrm{c} \in F$, we initialize an empty $\Theta^{\mathrm{c}\,r}_U-$tree. This tree is different from $\Theta^{\mathrm{c}\,r}_L-$tree and the structure of its leaves is similar to $(\Theta - \Lambda)^r_U-$tree. That is, there are $2n$ leaves in the tree, associated to the regular and delayed tasks. Each leaf of the $\Theta^{\mathrm{c}\,r}_U-$tree holds the parameters for the energy and latest energy envelope, respectively defined by (3) and (5), as well as (57) which is computed for $\Theta = \{i\}$. For an inner node w of the $\Theta^{\mathrm{c}}-$tree, (57) is computed for the set of all tasks which are included in the subtree rooted at w. This computation is done recursively as below.

$$\mathrm{Env}'^{\mathrm{c}\,k}_w = \min_{0 \leq j \leq k} (\mathrm{Env}'^{\mathrm{c}\,j}_{\mathrm{right}(w)} - \mathrm{e}^{k-j}_{\mathrm{left}(w)}, \mathrm{Env}'^{\mathrm{c}\,k}_{\mathrm{left}(w)}, \mathrm{Env}'^{\mathrm{c}(k-1)}_w) \tag{59}$$

$\Theta^{\mathrm{c}\,r}_U-$tree develops by processing all $\mathrm{prec}[i, b]$ in non-increasing order. The adjustment should be done only if $\mathrm{c}_i = \mathrm{c}$. If so, at iterating each $\mathrm{prec}[i, b] = \mathrm{est}_j$ for

Algorithm 8: Minlct(tree,bound,c,$\mathcal{C}$,k,b)

```
1  $v \leftarrow$ root;
2  e[0...$k$] $\leftarrow$ **0** ;
3  minEnv$^c$ $\leftarrow (\mathcal{C} - \mathrm{c})$.bound;
4  $k \leftarrow k - b$;
5  while $v$ is not a leaf do
6      branchLeft $\leftarrow$ false;
7      for $j = 0, ..., k$ do
8          if $\mathrm{Env}'^{c,j}_{\mathrm{left(v)}} - \mathrm{e}[k-j] <$minEnv$^c$ then
9              $v \leftarrow \mathrm{left(v)}$;
10             branchLeft $\leftarrow$ true;
11             break;
12     if not branchLeft then
13         $v \leftarrow \mathrm{right(v)}$;
14         for $j = 0, ..., k$ do
15             $\mathrm{e}'[j] \leftarrow \max_{0 \le i \le j}(\mathrm{e}[i] + \mathrm{e}^{j-i}_{\mathrm{left(v)}}, \mathrm{e}'[j-1])$
16         $\mathrm{e} \leftarrow \mathrm{e}'$;
17 return $v$;
```

some $j \in \mathcal{I}$, all the tasks which cannot start before j are scheduled in the $\Theta^{\mathrm{c}\,r}_{U}-$tree by adding their regular version to Θ^0 and their delayed version to Θ^1.

Let $\Omega^0 \cup \Omega^1$ be the candidate subset which can give the strongest adjustment in (55). Once all the tasks which are eligible to be in $\Omega^0 \cup \Omega^1$ are scheduled in the tree, it is time to identify Ω^0 and Ω^1 and compute the value of the second component of the right side of (55). We proceed to compute Ω^0 and Ω^1 in two steps.

The first step is to compute the minimum of $\max(\mathrm{lct}^0_{\mathcal{I}(\Omega^0)}, \mathrm{lct}^1_{\mathcal{I}(\Omega^1)})$ (or *minlct*) for all Ω^0 and Ω^1 that satisfy (55). In the algorithm 8, we consider this idea by taking advantage of the new variant of the latest energy envelope to locate such a node.

The second step is to compute the envelope of the tasks which is done in the algorithm 9 . This is similarly done by dividing the tasks in two groups $\alpha(j, \mathrm{c})$ and $\beta(j, c)$, for the tasks that finish at or after minlct and the tasks which finish before minlct. From the sets $\alpha(j, \mathrm{c})$ and $\beta(j, c)$, the latest energy envelop in the $\Theta^{\mathrm{c}\,r}_{U}-$tree is computed with

$$\mathrm{Env}(j, c) = \mathrm{Env}_{\alpha(j,\mathrm{c})} - \mathrm{e}_\beta \tag{60}$$

where e_β and Env_α are computed in a bottom-up manner, starting from the located task responsible for minlct, and by taking into account that at most r tasks are delayed. Once the envelope is computed, the value which makes the strongest update is adjusted by taking the minimum between the envelope computed in the current iteration and the preceding iterations (line 12 of the algorithm 7). Ultimately, if the task can not start before est_j, it gets scheduled itself.

Finally, it must be mentioned that the same complexity for filtering latest completion times is maintained and the argument is similar.

Algorithm 9: EnvelopeForUpperBound(v,tree,bound,c,k)

```
$e_\alpha \leftarrow [e_v^0, e_v^1, 0, ..., 0]$; // $e_\alpha$ includes $k$ - 1 entries 0
$Env'_\alpha \leftarrow [Env_v^0, Env_v^1, \infty, ..., \infty]$; // $Env_\alpha$ includes $k$ - 1 entries $\infty$
$e_\beta \leftarrow \mathbf{0}$ ;
while $v$ is not the root do
    if $v$ is a right child then
        $e'_\beta \leftarrow \mathbf{0}$ ;
        for $j = 0, ..., k$ do
            $e'_\beta[j] \leftarrow \max_{0 \le i \le j}(e_\beta[i] + e^{j-i}_{\mathrm{sibling}(v)}, e'_\beta[j-1])$
        $e_\beta \leftarrow e'_\beta$;
    else
        $Env''_\alpha \leftarrow \mathbf{0}$ ;
        $e'_\alpha \leftarrow \mathbf{0}$ ;
        for $j = 0, ..., k$ do
            $Env''_\alpha[j] \leftarrow \min(\min_{0 \le i \le j}(Env'^{i}_{\mathrm{sibling}(v)} - e_\alpha[j-i]), Env'_\alpha[j], Env''_\alpha[j-1])$;
            $e'_\alpha[j] \leftarrow \max(\max_{0 \le i \le j}(e^i_{\mathrm{sibling}(v)} + e_\alpha[j-i]), e'_\alpha[j-1])$;
    $Env'_\alpha \leftarrow Env''_\alpha$;
    $e_\alpha \leftarrow e'_\alpha$;
    $v \leftarrow \mathrm{parent}(v)$;
return $\min_{0 \le i \le k}(Env'_\alpha[k-i] - e_\beta[i])$;
```

5 Experiments

The experiments were carried out on a 2.0 GHz Intel Core i5, with Choco version 3.3.1. We tested our algorithms against the BL suite of the RCPSP instances [2]. This benchmark consists of 40 highly cumulative instances with either 20 or 25 tasks, subject to precedence constraints, to be executed on several resources. We minimize the makespan. For the delay attributes associated to every task i, we uniformly generated random numbers in $[0, 2 \cdot p_i^0]$. We used three different heuristics: Lexicographic, DomOverWDeg [6], and Impact Based Search [17]. For these three heuristics, the figures 5, 6 and 7 respectively illustrate a logarithmic scale representation of the number of backtracks and the elapsed time measurements when a combination of time-tabling and overload checking or time-tabling and Edge-Finding are implemented. For the time-tabling algorithm, we used the same implementation as in Derrien et al. [7] that we obtained from the authors.

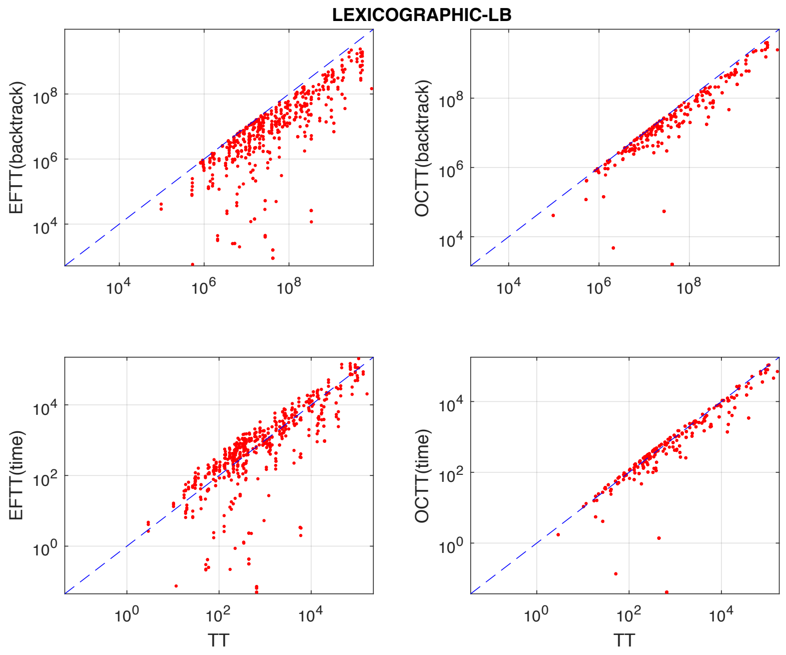


Fig. 5 The logarithmic scale graphs as the results of running time-tabling and overload checking (denoted OC-TT) or time-tabling and Edge-Finding(denoted EF-TT) with lexicographic heuristics in terms of backtrack numbers as well as elapsed times. The horizontal axis corresponds to the time-tabling and the vertical axis corresponds to overload checking or Edge-Finding.

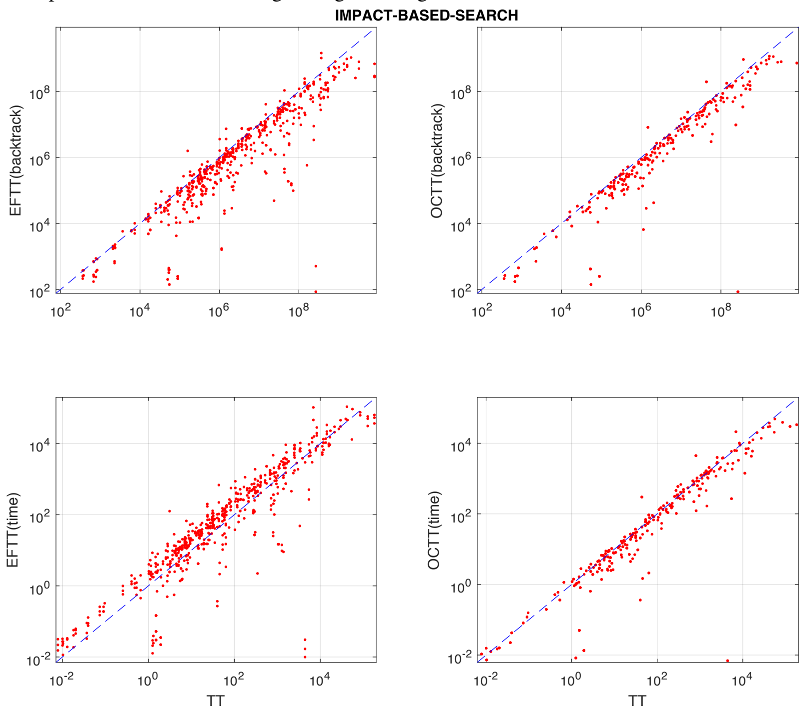


Fig. 6 The logarithmic scale graphs as the results of running time-tabling and overload checking (denoted OC-TT) or time-tabling and Edge-Finding(denoted EF-TT) with impact based search heuristics in terms of backtrack numbers as well as elapsed times. The horizontal axis corresponds to the time-tabling and the vertical axis corresponds to overload checking or Edge-Finding.

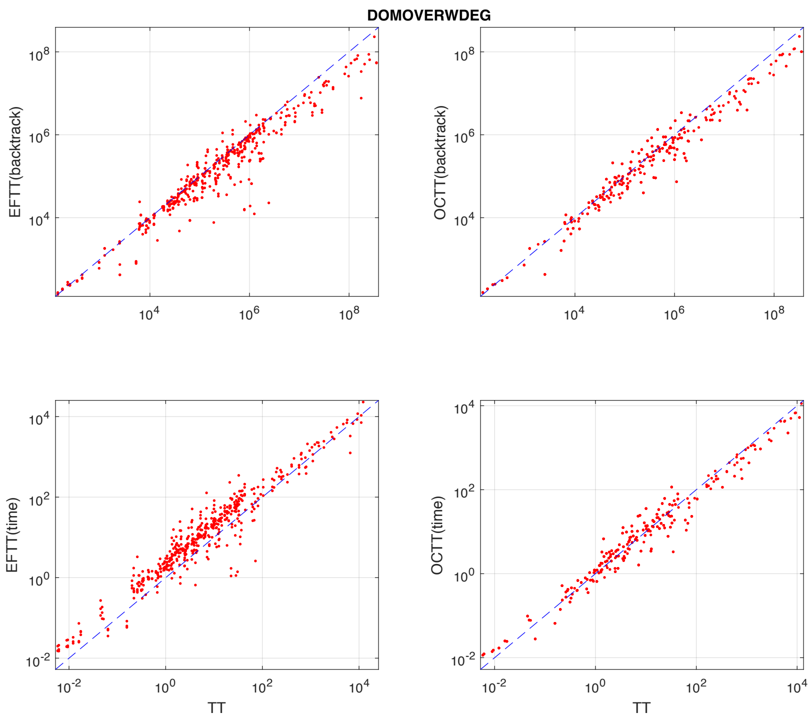


Fig. 7 The logarithmic scale graphs as the results of running time-tabling and overload checking (denoted OC-TT) or time-tabling and Edge-Finding(denoted EF-TT) with domoverwdeg heuristics in terms of backtrack numbers as well as elapsed times. The horizontal axis corresponds to the time-tabling and the vertical axis corresponds to overload checking or Edge-Finding.

As the graphs verify, our algorithms lead to fewer backtracks. The results are much more significant for the Edge-Finding. This is due to the fact that the Edge-Finding filters the domains while the overload checking only triggers backtracks. It appears that the heuristic chosen for solving the problem also affects the results. Thereby, we selected the lexicographic heuristic as one of our heuristics so that the heuristic chosen does not impinge our objective, which is proving that the combination of our algorithms with the time tabling provides a stronger filtering. The lexicographic heuristic is certainly not the best heuristic to solve scheduling problems, but it allows seeing how much pruning a new filtering algorithm achieves. The combination of our algorithms with time tabling improves the resolution times for many instances. This fact can differ from one heuristic to another. Overall, among the state of the art heuristics, IMPACT-BASED-SEARCH performs more efficiently, as it leads to fewer backtracks and faster computation times for many instances.

We also compared the implementation of our Edge-Finding algorithm with a conjunction of n CUMULATIVE constraints, as described in section 2.2. The performances for the conjunction of n CUMULATIVE constraints are so much worse in terms of time that we omit to report them.

6 Conclusion

We adapted the state of the art algorithms for overload checking and Edge-Finding in robust cumulative scheduling problems. The experimental results demonstrate a stronger filtering when our algorithms are combined with time-tabling.

Acknowledgments

The authors would like to thank Alban Derrien for providing his implementation of the time tabling algorithm [7].

References

1. Christian Artigues and Pierre Lopez. Energetic reasoning for energy-constrained scheduling with a continuous resource. *Journal of Scheduling*, 18(3):225–241, 2015.
2. Philippe Baptiste and Claude Le Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints*, 5(1-2):119–139, 2000.
3. Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer Science & Business Media, 2012.
4. N. Beldiceanu and M. Carlsson. A new multi-resource cumulatives constraint with negative heights. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP 2002)*, pages 63–79, 2002.
5. N. Beldiceanu, M. Carlsson, and E. Poder. New filtering for the cumulative constraint in the context of non-overlapping rectangles. In *Proceedings of the 5th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CPAIOR 2008)*, pages 21–35, 2008.
6. Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *ECAI*, volume 16, page 146, 2004.
7. Alban Derrien, Thierry Petit, and Stéphane Zampelli. A declarative paradigm for robust cumulative scheduling. In *Principles and Practice of Constraint Programming*, pages 298–306. Springer, 2014.
8. Hamed Fahimi and Claude-Guy Quimper. Linear-time filtering algorithms for the disjunctive constraint. In *AAAI*, pages 2637–2643, 2014.
9. Steven Gay, Renaud Hartert, and Pierre Schaus. Simple and scalable time-table filtering for the cumulative constraint. In *Principles and Practice of Constraint Programming*, pages 149–157. Springer, 2015.
10. Roger Kameugne, Laure Pauline Fotso, Joseph Scott, and Youcheu Ngo-Kateu. A quadratic edge-finding filtering algorithm for cumulative resource constraints. *Constraints*, 19(3):243–269, 2014.
11. A. Letort, N. Beldiceanu, and M. Carlsson. A scalable sweep algorithm for the cumulative constraint. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP 2012)*, pages 439–454, 2012.
12. Arnaud Letort, Nicolas Beldiceanu, and Mats Carlsson. A scalable sweep algorithm for the cumulative constraint. In *Principles and Practice of Constraint Programming*, pages 439–454. Springer, 2012.
13. C. Le Pape. *Des systèmes d'ordonnancement flexibles et opportunistes*. PhD thesis, Université Paris IX, 1988.
14. L. Mercier and P. Van Hentenryck. Edge finding for cumulative scheduling. *INFORMS Journal on Computing*, 20(1):143–153, 2008.
15. WPM Wim Nuijten. *Time and resource constrained scheduling: a constraint satisfaction approach*. PhD thesis, Technische Universiteit Eindhoven, 1994.
16. P. Ouellet and C.-G. Quimper. Time-table-extended-edge-finding for the cumulative constraint. In *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming (CP 2013)*, pages 562–577, 2013.
17. Philippe Refalo. Impact-based search strategies for constraint programming. In *Principles and Practice of Constraint Programming–CP 2004*, pages 557–571. Springer, 2004.

18. Joseph Scott. Filtering algorithms for discrete cumulative resources. 2010.
19. P. Vilím. Edge finding filtering algorithm for discrete cumulative resources in o(kn log n). In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming*, pages 802–816, 2009.
20. Petr Vilím. Max energy filtering algorithm for discrete cumulative resources. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 294–308. Springer, 2009.
21. Armin Wolf and Gunnar Schrader. {\ cal O}(n\ log n) overload checking for the cumulative constraint and its application. In *Declarative Programming for Knowledge Management*, pages 88–101. Springer, 2005.

Noname manuscript No.
(will be inserted by the editor)

Overload Checking and Edge-Finding for Robust Cumulative Scheduling

Hamed Fahimi · Claude-Guy Quimper

Received: date / Accepted: date

Abstract We present two new filtering algorithms for the FlexC constraint, a constraint that models cumulative scheduling problems where up to r tasks can be delayed while keeping the schedule valid. We adapt the overload checking and Edge-Finding filtering rules for this framework. It turns out that the complexities of the state of the art algorithms for these techniques are maintained, when the number of delayed tasks r is constant. The experimental results verify a stronger filtering for these methods when used in conjunction with time-tabling. Furthermore, the computation times show a faster filtering for specific heuristics for many instances.

1 Introduction

Project scheduling problems deal with allocating scarce resources over time in order to perform jobs. This context arises in a great variety of environments, such as in the industry to plan operations. *Constraint programming* is a powerful methodology to solve large scale and practical scheduling problems. Filtering techniques are being developed and improved over the past years in constraint-based scheduling. The prominency of filtering algorithms lies on their power to shrink the search tree by excluding values from the domains which do not yield a feasible solution. *Robust project scheduling* is concerned with unruly environments which are subject to disruptive behaviours that are caused by undesirable factors. In such contexts, the execution of tasks takes longer than expected. Inevitably, re-planning is necessary, however we intend to focus on robust scheduling.

Claude-Guy Quimper
Tel.: +1-418-656-2131 ext. 2099
Fax: -
E-mail: Claude-Guy.Quimper@ift.ulaval.ca

Hamed Fahimi
Tel: +1-418-656-2131 ext. 4799
E-mail: hamed.fahimi.1@ulaval.ca

This work is based on the framework provided by Derrien et al. [?] for robust cumulative scheduling, where they assume that for a set of n tasks $\mathcal{I}$, at most r of them can be delayed without the requirement to reschedule the alternative tasks. The considered stochastic framework can capture practical problems, such as the crane assignment problem [?]. In this problem, the planner needs a schedule which still respects the deadlines in case at most r tasks are delayed. Derrien et.al present the adaption of time-tabling algorithm. Even though they define the paradigm for r delayed tasks, they focus on the case $r = 1$. This paper adapts the overload checking [?] and Edge-Finding [?] for any $r > 0$ with this new framework.

The paper is structured as follows. Section 2 describes the terms and notations which are most frequently used throughout the paper. It also presents the CUMULATIVE constraint as well as the FLEXC constraint as a variation of CUMULATIVE constraint in robust contexts. Moreover, it surveys the filtering techniques time-tabling, overload checking as well as Edge-Finding. Section 3 presents the robust algorithm of overload checking. Section 4 is devoted to the Edge-Finding algorithm for filtering the domains of the starting time variables. Section 5 presents and discuss the experiments. In section 6 we conclude.

2 Preliminaries and the general framework

We tackle the scheduling problem for a set of tasks $\mathcal{I} = \{1, ..., n\}$ to be executed on a resource of finite capacity $\mathcal{C}$. Presuming the time is discrete, that is the values of the attributes are integers, every task $i \in \mathcal{I}$ is characterized by an earliest starting time est_i, a latest completion time lct_i^0, a capacity c_i and a processing time p_i^0. The *latest starting time* (lst_i) of a task is the largest date at which it can start executing and the *earliest completion time* (ect_i) of a task is the earliest date at which it can cease to execute. These two values are computed with $\mathrm{lst}_i = \mathrm{lct}_i^0 - p_i^0$ and $\mathrm{ect}_i = \mathrm{est}_i + p_i^0$, respectively. The *starting time* S_i of a task is the time point at which it starts executing. The starting times are the unknowns of a scheduling problem. The *energy* of i, denoted e_i, is the sum over time of the capacity of i, computed by $\mathrm{e}_i = \mathrm{c}_i\, p_i^0$. One may generalize these notions for an arbitrary subset $\Theta \subseteq \mathcal{I}$ of tasks as follows

$$\mathrm{est}_\Theta = \min\{\mathrm{est}_i : i \in \Theta\} \tag{1}$$

$$\mathrm{lct}_\Theta^0 = \max\{\mathrm{lct}_i^0 : i \in \Theta\} \tag{2}$$

$$\mathrm{e}_\Theta = \sum_{i \in \Theta} \mathrm{e}_i \tag{3}$$

For an empty set we assume that $\mathrm{est}_\emptyset = \infty$ and $\mathrm{lct}_\emptyset^0 = -\infty$.

Vilím [?] introduces the concept of the *earliest energy envelope* and the *latest energy envelope* of Θ, which are respectively defined by

$$\mathrm{Env}_\Theta = \max_{\Omega \subseteq \Theta}(\mathcal{C}\,\mathrm{est}_\Omega + \mathrm{e}_\Omega) \tag{4}$$

$$\mathrm{Env}'_\Theta = \min_{\Omega \subseteq \Theta}(\mathcal{C}\,\mathrm{lct}_\Omega^0 - \mathrm{e}_\Omega) \tag{5}$$

For $\Omega \subseteq \Theta$, Env_Θ provides the aggregation of the energy which is required to fully use the resource up to time est_Ω and the energy of executing the task in Ω. Note that a subset $\Omega \subset \Theta$ might give the maximum, if $\mathrm{est}_\Omega > \mathrm{est}_\Theta$. A similar reasoning for Env'_Θ holds.

It is convenient to compute the latest starting time and earliest completion time of a set of tasks Θ. Thanks to (4) and (5), a lower bound for the earliest completion time and an upper bound for the latest starting time of Θ are respectively obtained by $\mathrm{ect}_\Theta = \lceil \mathrm{Env}_\Theta / \mathcal{C} \rceil$ and $\mathrm{lst}_\Theta = \lfloor \mathrm{Env}'_\Theta / \mathcal{C} \rfloor$ [**?**].

Due to some considerable uncertainties, one can not absolutely trust the task durations, as the processing of tasks can take longer than expected. Such a situation entails the assignment of an attribute d_i known as the *delay duration* to each task i. From that, a task i has two extra parameters, the delayed processing time $p_i^1 = p_i^0 + d_i$ and the delayed latest completion time $\mathrm{lct}_i^1 = \mathrm{lct}_i^0 + d_i$. Furthermore,

$$\mathrm{lct}_\Theta^1 = \max\{\mathrm{lct}_i^1 \mid i \in \Theta\} \tag{6}$$

We refer to the tasks when they are delayed or not delayed with particular symbols. A task i that is not delayed is called *regular* and it is denoted i^0 and a task i that is delayed is denoted i^1. Throughout this paper, we associate a regular task with 0 and a delayed task with 1. For instance, the set $\{A^0, B^1, C^0, D^1\}$ is a set of tasks where A and C are not delayed and B and D are delayed. $\mathcal{I}^0 = \{i^0 \mid i \in \mathcal{I}\}$ refers to the regular tasks from $\mathcal{I}$ and $\mathcal{I}^1 = \{i^1 \mid i \in \mathcal{I}\}$ refers to the delayed tasks from $\mathcal{I}$. Moreover, for $i^1 \in \mathcal{I}^1$, the processing time p_i^1 is considered and for $i^0 \in \mathcal{I}^0$, the processing time p_i^0 is considered. For $i \in \mathcal{I}$ and $b \in \{0,1\}$, we define $\mathrm{e}_{i^b} = p_i^b \, \mathrm{c}_i$. For a subset $\Theta \subseteq \mathcal{I}^0 \cup \mathcal{I}^1$, we define

$$\mathrm{e}_\Theta = \sum_{i^b \in \Theta} \mathrm{e}_{i^b} \tag{7}$$

For a subset $\Theta \subseteq \mathcal{I} \cup \mathcal{I}^0 \cup \mathcal{I}^1$,

$$\mathcal{I}(\Theta) = \{i \in \mathcal{I} \mid i^0 \in \Theta \vee i^1 \in \Theta \vee i \in \Theta\} \subseteq \mathcal{I} \tag{8}$$

Note that $\mathcal{I}(\Theta)$ is the set of tasks from $\mathcal{I}$, whatsoever. That is, the delay status of the tasks is not specified in $\mathcal{I}(\Theta)$.

2.1 CUMULATIVE constraint

The CUMULATIVE constraint models a relationship between a scarce resource and the tasks which are to be processed on the resource. Let $S_1, ..., S_n$ denote the starting times of the tasks as the decision variables of the problem, where the domain of $S_i, i \in \mathcal{I}$, is the interval $[\mathrm{est}_i, \mathrm{lst}_i]$. The constraint CUMULATIVE$([S_1, \ldots, S_n], [p_1^0, \ldots, p_n^0], [\mathrm{c}_1, \ldots, \mathrm{c}_n], \mathcal{C})$ holds if and only if

$$\forall t : \sum_{S_i \le t < S_i + p_i^0} \mathrm{c}_i \le \mathcal{C} \tag{9}$$

The decision problem related to the CUMULATIVE constraint is called the *cumulative scheduling problem* (CuSP) [**?**].

2.2 Robust cumulative constraint

Derrien et.al [**?**] introduce the Robust Cumulative Problem of order r (RCuSP^r) by integrating the notion of robustness to CuSP. According to this framework, a set $\Theta^1 \subseteq \mathcal{I}^1$ of at most $r \geq 1$ tasks can be delayed up to their associated delay attribute without shifting the position of other tasks. A solution to RCuSP^r satisfies

$$\forall t, \forall \Theta^1 \subseteq \mathcal{I}^1, |\Theta^1| \leq r : \sum_{i \in \mathcal{I}: S_i \leq t < S_i + p_i^0} \mathrm{c}_i + \sum_{j^1 \in \Theta^1 : S_j + p_j^0 \leq t < S_j + p_j^1} \mathrm{c}_j \leq \mathcal{C} \quad (10)$$

The first summation in constraint (10) adds the capacities of all tasks executing at time t and the second summation adds the capacities of at most r tasks whose delayed part intersects with time t. We refer to the constraint (10) by
$\textsc{FlexC}([S_1, \ldots, S_n], [p_1^0, \ldots, p_n^0], [d_1, \ldots, d_n], [\mathrm{c}_1, \ldots, \mathrm{c}_n], \mathcal{C}, r)$.
Let

$$1(x) = \begin{cases} 1 \text{ if } x \text{ is true} \\ 0 \text{ if } x \text{ is false} \end{cases} \quad (11)$$

The following relation indicates that this problem considers $\binom{n}{r}$ scenarios where r tasks among n are delayed and the CUMULATIVE constraint holds no matter which of r tasks are delayed [**?**].

$$\textsc{FlexC}([S_1, \ldots, S_n], [p_1^0, \ldots, p_n^0], [d_1, \ldots, d_n], [\mathrm{c}_1, \ldots, \mathrm{c}_n], \mathcal{C}, r) \iff$$

$$\bigwedge_{\substack{\Theta^1 \subseteq \mathcal{I}^1 \\ |\Theta^1| = r}} \textsc{Cumulative}([S_1, ..., S_n], [p_1^{1(1 \in \Theta^1)}, ..., p_n^{1(n \in \Theta^1)}], [\mathrm{c}_1, ..., \mathrm{c}_n], \mathcal{C}) \quad (12)$$

The algorithms that we adapt to cover delayed tasks efficiently emulates the state of the art algorithms on the conjunction of CUMULATIVE constraints (12).

2.3 Time-Tabling

Time-Tabling is a filtering technique which maintains a minimal resource consumption at each time t, which allows the solver to restrict the domains of other tasks by preventing them from execution at times that would lead to the over-consumption of the resource. For a task $i \in \mathcal{I}$, if $\mathrm{lst}_i < \mathrm{ect}_i$, the time window $[\mathrm{lst}_i, \mathrm{ect}_i)$ is called the *compulsory part* of task i. That is, if it exists, a compulsory part is a time window in which a task certainly executes. Let $f(t, \Theta) = \sum_{i \in \Theta | t \in [\mathrm{lst}_i, \mathrm{ect}_i)} c_i$ denote the amount of energy that is consumed by the tasks of $\Theta \subseteq \mathcal{I}$ for which t lies in their compulsory parts. $f(t, \Theta)$ provides a lower bound on the resource consumption at time t. The time tabling rules are as follows, where the rule (13) filters the earliest starting times and the rule (14) filters the latest completion times. The left side of the implications corresponds to the detection test and the right side corresponds to the adjustment.

$$(\mathrm{c}_i + f(t, \mathcal{I} \setminus \{i\}) > \mathcal{C}) \wedge (t < \mathrm{ect}_i) \Rightarrow \mathrm{est}_i > t \quad (13)$$

$$(\mathrm{c}_i + f(t, \mathcal{I} \setminus \{i\}) > \mathcal{C}) \wedge (\mathrm{lst}_i \leq t) \Rightarrow \mathrm{lct}_i^0 < t - p_i \quad (14)$$

Notice that after applying the rules, the same task might get filtered further with respect to another time t.

Several algorithms apply the time-tabling rules [?,?,?,?,?,?]. Derrien et al., adapt the sweep algorithm proposed by Arnaud Letort, et.al. in [?] for the FLEXC constraint. It ensures that even if up to r tasks are delayed, no time tabling rules are triggered.

2.4 Overload checking

The *overload checking* rule ensures that the energy of a set of tasks $\Theta \subseteq \mathcal{I}$ does not overflow with the capacity of the resource available within the window where the tasks must execute. That is,

$$\forall \Theta \subseteq \mathcal{I} : \mathrm{e}_\Theta \leq C(\mathrm{lct}^0_\Theta - \mathrm{est}_\Theta) \tag{15}$$

Overload checking provides a necessary condition for the existence of a solution. Wolf and Schrader introduced an algorithm for overload checking which runs in $O(n \log(n))$ [?]. Vilím [?] introduced the idea of $\Theta-$tree, which is a balanced binary tree, and by taking advantage of that he also presented an algorithm in $O(n \log(n))$ for the overload checking. Recently, Fahimi and Quimper [?] proposed a linear time algorithm for this test.

2.5 Edge-Finding

This is a filtering technique which was firstly introduced by Baptiste et.al, [?]. The idea of Edge Finding for filtering the lower bounds of starting times is to detect a subset of tasks $\Theta \subset \mathcal{I}$ and a task $i \in \mathcal{I} \setminus \Theta$ such that, in any solution, all the tasks of Θ complete before i completes. This property is denoted $\Theta \prec i$ and it is called a *precedence*. The following rules capture the bounding techniques of Edge-Finding:

$$\forall \Theta, \forall i \notin \Theta, C(\mathrm{lct}^0_\Theta - \mathrm{est}_{\Theta \cup \{i\}}) < \mathrm{e}_{\Theta \cup \{i\}} \Rightarrow \Theta \prec i \tag{16}$$

$$\forall \Theta, \forall i \notin \Theta, C(\mathrm{lct}^0_{\Theta \cup \{i\}} - \mathrm{est}_\Theta) < \mathrm{e}_{\Theta \cup \{i\}} \Rightarrow i \prec \Theta \tag{17}$$

The idea can be deduced from the overload checking, in the sense that if the scheduling of task i at its release time causes an overload in the interval where Θ is allowed to execute, then Θ has to precede i. A symmetric reasoning applies for the rule (17). Once the appropriate Θ and i are identified, the temporal time bounds must be adjusted. For $\Omega \subseteq \Theta$, there are e_Ω units of energy within the interval $[\mathrm{est}_\Omega, \mathrm{lct}^0_\Omega)$. Let

$$\mathrm{rest}(\Omega, c_i) = \mathrm{e}_\Omega - (\mathcal{C} - \mathrm{c}_i)(\mathrm{lct}^0_\Omega - \mathrm{est}_\Omega) \tag{18}$$

Baptiste et al., [?] proved that the subsets which have enough energy to prevent concurrent scheduling of i are those for which $\mathrm{rest}(\Omega, c_i) > 0$ and if so, $\mathrm{est}_\Omega + \lceil \mathrm{rest}(\Omega, c_i) \rceil$

provides a lower bound for est_i and $\mathrm{lct}^0_\Omega - \lceil \mathrm{rest}(\Omega, c_i) \rceil$ provides an upper bound for lct^0_i. The following relations for the new bounds are deduced:

$$\mathrm{est}_i \leftarrow \max(\mathrm{est}_i, \max_{\substack{\emptyset \neq \Omega \subseteq \Theta \\ \mathrm{e}_\Omega > (\mathcal{C} - \mathrm{c}_i)(\mathrm{lct}^0_\Omega - \mathrm{est}_\Omega)}} \{\mathrm{est}_\Omega + \left\lceil \frac{\mathrm{rest}(\Omega, c_i)}{\mathrm{c}_i} \right\rceil\}) \tag{19}$$

$$\mathrm{lct}^0_i \leftarrow \min(\mathrm{lct}^0_i, \min_{\substack{\emptyset \neq \Omega \subseteq \Theta \\ \mathrm{e}_\Omega > (\mathcal{C} - \mathrm{c}_i)(\mathrm{lct}^0_\Omega - \mathrm{est}_\Omega)}} \{\mathrm{lct}^0_\Omega - \left\lceil \frac{\mathrm{rest}(\Omega, c_i)}{\mathrm{c}_i} \right\rceil\}) \tag{20}$$

Several algorithms exists for Edge-Finding [**?**,**?**]. Vilím introduces an algorithm which admits a running time of $O(kn \log(n))$, where k signifies the number of distinct capacities associated to the tasks in $\mathcal{I}$ [**?**] . Roger Kameugne et. al, [**?**] present an Edge-Finding algorithm in $O(n^2)$. They empirically prove that their algorithm is substantially faster than Vilím's Edge-Finding.

3 Robust overload checking

The objective of this section is to adapt the overload checking algorithm introduced in [**?**] for the FLEXC$([S_1, \ldots, S_n], [p^0_1, \ldots, p^0_n], [d_1, \ldots, d_n], [\mathrm{c}_1, \ldots, \mathrm{c}_n], \mathcal{C}, r)$ constraint. In section 3.1 we establish the generic form of the robust overload checking rule and illustrate it with an example. Section 3.2 introduces the notion of earliest energy envelope of a set of tasks in the robust context. Section 3.3 recasts the robust overload checking rule in terms of the robust energy envelope. Afterwards, in section 3.4 we propose an extended data structure which enables us to handle the tasks when a certain number of them are delayed. Finally, in section 3.5 we take advantage of the introduced data structure to present the robust overload checking algorithm. Moreover, we trace few steps of the algorithms for the example provided. This section finishes with a discussion on the time complexity of the algorithm.

3.1 The general form of robust overload checking rule

Let $\Theta^0 \subseteq \mathcal{I}^0, \Theta^1 \subseteq \mathcal{I}^1$, such that $|\Theta^1| \leq r$ and $\mathcal{I}(\Theta^0) \cap \mathcal{I}(\Theta^1) = \emptyset$. The overload checking triggers a failure if

$$\mathcal{C}(\max(\mathrm{lct}^0_{\mathcal{I}(\Theta^0)}, \mathrm{lct}^1_{\mathcal{I}(\Theta^1)}) - \mathrm{est}_{\mathcal{I}(\Theta^0 \cup \Theta^1)}) < \mathrm{e}_{\Theta^0} + \mathrm{e}_{\Theta^1} \tag{21}$$

Example 1 Consider the table 1 which corresponds to a set of tasks $\mathcal{I} = \{A, B, C, D, E\}$ that must execute on a resource of capacity $\mathcal{C} = 4$ in the context where at most $r = 2$ tasks are allowed to delay. Let $\Theta = \{A, B, C, D\} \subseteq \mathcal{I}$. If $\Theta^0 = \{A^0, C^0\}$ and $\Theta^1 = \{B^1, D^1\}$, the overload checking returns a failure, for

$$100 = 4(25 - 0) < (10 + 12) + (36 + 44) = 102$$

□

task	est	lct^0	p^0	d	c
A	1	20	5	1	2
B	4	23	7	2	4
C	0	14	4	0	3
D	0	21	8	3	4
E	9	26	2	1	1

Table 1 A set of tasks $\mathcal{I} = \{A, B, C, D, E\}$ to execute on a resource of capacity $\mathcal{C} = 4$. The overload checking fails according to (21) for $\Theta = \{A, B, C, D\} \subseteq \mathcal{I}, \Theta^0 = \{A^0, C^0\}$ and $\Theta^1 = \{B^1, D^1\}$.

3.2 Robust earliest energy envelope

One can generalize the notion of the earliest energy envelope when tasks can be delayed. Rather than computing the earliest energy envelope of a set Θ as in (4), we compute the earliest energy envelope of two sets $\Theta^0 \subseteq \mathcal{I}^0$ and $\Theta^1 \subseteq \mathcal{I}^1$. The tasks in Θ^0 can be regular while the tasks in Θ^1 can be delayed. The tasks that belong to both sets can either be regular or delayed but not both. The earliest energy envelope for the sets Θ^0 and Θ^1 in such a case is defined

$$\mathrm{Env}^r(\Theta^0, \Theta^1) = \max_{\substack{\Omega^0 \subseteq \Theta^0 \\ \Omega^1 \subseteq \Theta^1 \\ |\Omega^1| \leq r \\ \mathcal{I}(\Omega^0) \cap \mathcal{I}(\Omega^1) = \emptyset}} \left(\mathcal{C}\, \mathrm{est}_{\mathcal{I}(\Omega^0 \cup \Omega^1)} + \mathrm{e}_{\Omega^0} + \mathrm{e}_{\Omega^1}\right) \tag{22}$$

Once (22) is computed, a lower bound for the earliest completion time of $\Theta^0 \cup \Theta^1$ is obtained by $\lceil \mathrm{Env}^r(\Theta^0, \Theta^1)/\mathcal{C} \rceil$.

3.3 Robust overload checking rule in terms of robust energy envelope

We establish an equivalent criterion for the overload checking rule (21) in terms of $\mathrm{Env}^r(\Theta^0, \Theta^1)$. Let $T = \{\mathrm{lct}_i^0 : i \in \mathcal{I}\} \cup \{\mathrm{lct}_i^1 : i \in \mathcal{I}\}$ be the set of all latest completion time and delayed latest completion time points. For $t \in T$, let $\mathrm{Lcut}^0(t) = \{i^0 \in \mathcal{I}^0 : \mathrm{lct}_i^0 \leq t\}$ be the *regular left cut* of t and $\mathrm{Lcut}^1(t) = \{i^1 \in \mathcal{I}^1 : \mathrm{lct}_i^1 \leq t\}$ be the *delayed left cut* of t. The former signifies the set of regular tasks which terminate no later than t and the latter refers to the set of delayed tasks which terminate no later than t.

Lemma 1 *The overload checking fails according to the rule (21) if and only if for some $t \in T$*

$$\mathrm{Env}^r(\mathrm{Lcut}^0(\mathrm{t}), \mathrm{Lcut}^1(\mathrm{t})) > \mathcal{C} \cdot \mathrm{t}$$

Proof Consider $\Theta^0 \subseteq \mathcal{I}^0$ and $\Theta^1 \subseteq \mathcal{I}^1$ with $|\Theta^1| \leq r$ and $\mathcal{I}(\Theta^0) \cap \mathcal{I}(\Theta^1) = \emptyset$ as the subsets for which (21) implies that the overload checking fails and let $\max(\mathrm{lct}^0_{\mathcal{I}(\Theta^0)}, \mathrm{lct}^1_{\mathcal{I}(\Theta^1)}) = \mathrm{lct}^0_{\mathcal{I}(\Theta^0)}$ in (21). If j corresponds to a task $j \in \mathcal{I}(\Theta^0)$ for which $\mathrm{lct}_j^0 = \mathrm{lct}^0_{\mathcal{I}(\Theta^0)}$ and by setting $t = \mathrm{lct}_j^0$, the assumption for the selected task j implies $\Theta^0 \subseteq \mathrm{Lcut}^0(t)$ and $\Theta^1 \subseteq \mathrm{Lcut}^1(t)$. From (21) it follows that

$$\mathcal{C} \cdot t = \mathcal{C} \cdot \mathrm{lct}^0_{\mathcal{I}(\Theta^0)} < \mathcal{C} \cdot \mathrm{est}_{\mathcal{I}(\Theta^0 \cup \Theta^1)} + \mathrm{e}_{\Theta^0} + \mathrm{e}_{\Theta^1} \leq \mathrm{Env}^r(\mathrm{Lcut}^0(\mathrm{t}), \mathrm{Lcut}^1(\mathrm{t}))$$

A similar reasoning holds if $\max(\mathrm{lct}^0_{\mathcal{I}(\Theta^0)}, \mathrm{lct}^1_{\mathcal{I}(\Theta^1)}) = \mathrm{lct}^1_{\mathcal{I}(\Theta^1)}$.

Now, assume that for some $t \in T$, $\mathrm{Env}^r(\mathrm{Lcut}^0(t), \mathrm{Lcut}^1(t)) > \mathcal{C} \cdot t$. Let $\Theta^0 \subseteq \mathrm{Lcut}^0(t)$, $\Theta^1 \subseteq \mathrm{Lcut}^1(t)$ be the subsets for which

$$\mathrm{Env}^r(\mathrm{Lcut}^0(t), \mathrm{Lcut}^1(t)) = \mathcal{C}\, \mathrm{est}_{\mathcal{I}(\Theta^0 \cup \Theta^1)} + \mathrm{e}_{\Theta^0} + \mathrm{e}_{\Theta^1}$$

where $\left|\Theta^1\right| \leq r$ and $\mathcal{I}(\Theta^0) \cap \mathcal{I}(\Theta^1) = \emptyset$. Therefore,

$$\mathcal{C}{\cdot}\max(\mathrm{lct}^0_{\mathcal{I}(\Theta^0)}, \mathrm{lct}^1_{\mathcal{I}(\Theta^1)}) \leq \mathcal{C}{\cdot}t < \mathrm{Env}^r(\mathrm{Lcut}^0(t), \mathrm{Lcut}^1(t)) = \mathcal{C}\, \mathrm{est}_{\mathcal{I}(\Theta^0 \cup \Theta^1)} + \mathrm{e}_{\Theta^0} + \mathrm{e}_{\Theta^1}$$

which implies (21). □

In the following, we present a data structure from which $\mathrm{Env}^r(\Theta^0, \Theta^1)$ can be retrieved efficiently.

3.4 Θ^r_L-tree

Vilím introduces the concept of $\Theta-$tree [**?**], which is a balanced binary tree with one leaf for each task. In the $\Theta-$tree, the leaves, when enumerated from left to right, correspond to the tasks sorted in non-decreasing order of the earliest starting times. Each node holds the parameters of energy, defined by (3) as well as the earliest energy envelope, defined by (4). For a leaf v of the $\Theta-$tree, which corresponds to a task $i \in \mathcal{I}$, the energy and the earliest energy envelope are computed for $\Theta = \{i\}$ and for an inner node w of the $\Theta-$tree, these parameters are computed for the set of all tasks which are included in the subtree rooted at w. The parameters for w are computed in a bottom-up manner. That is, the parameters of every inner node in the $\Theta-$tree depend on the parameters of its children. The structure of the tree is important in the sense that it allows to compute the earliest energy envelope by updating the tree through a bottom-up traversal to the root and ultimately retrieving the earliest completion time of the set of tasks which are scheduled thus far. While maintaining the same structure for the $\Theta-$tree, we extend it to a tree, called Θ^r_L-tree, by adding new parameters to the nodes to handle the case where at most r tasks could be delayed. In the Θ^r_L-tree, two sets of task $\Theta^0 \subseteq \mathcal{I}^0$ and $\Theta^1 \subseteq \mathcal{I}^1$ affect the values of the parameters in the leaves and therefore in the entire tree. In what follows, the symbols $\mathrm{e}^0_v, \mathrm{Env}^0_v$ and $\mathrm{e}^1_v, \mathrm{Env}^1_v$ stand for the energy and the earliest energy envelope of a task v whether this task is regular or delayed. They are computed as

$$\mathrm{e}^k_v = \begin{cases} \mathrm{c}_v\, p^0_v & \text{if } (v^0 \in \Theta^0) \wedge (k = 0 \vee v^1 \notin \Theta^1) \\ \mathrm{c}_v\, p^1_v & \text{if } (v^1 \in \Theta^1) \wedge (k > 0) \\ 0 & \text{otherwise} \end{cases} \tag{23}$$

$$\mathrm{Env}^k_v = \begin{cases} \mathcal{C}\, \mathrm{est}_v + \mathrm{e}^k_v & \text{if } (v^0 \in \Theta^0) \vee (v^1 \in \Theta^1) \\ -\infty & \text{otherwise} \end{cases} \tag{24}$$

The superscript k stands for an upper bound on the number of delayed tasks in the leaf v. Since the task corresponding to v is either regular or delayed, only two cases for $k (k \in \{0, 1\})$ make sense. However, in order to make the computations of the energy and envelope symmetrical over all nodes of the tree, we suppose that $0 \leq k \leq r$ and since e_v^k and Env_v^k are the energy and earliest energy envelope when at most k tasks are delayed, for $k \geq 2$ in the leaves of the Θ_L^r-tree we necessarily have $\mathrm{e}_v^k = \mathrm{e}_v^1$ and $\mathrm{Env}_v^k = \mathrm{Env}_v^1$.

Let w be an internal node of the tree. The symbols e_w^r and Env_w^r stand for the energy and envelope of all tasks associated to the leaves that are descendants of w. Moreover, among these tasks, a most r tasks in Θ^1 can be delayed. If the left and right children of w are respectively denoted left(w) and right(w), it is proven [**?**] that

$$\mathrm{e}_w^0 = \mathrm{e}_{\mathrm{left}(w)}^0 + \mathrm{e}_{\mathrm{right}(w)}^0 \tag{25}$$

$$\mathrm{Env}_w^0 = \max(\mathrm{Env}_{\mathrm{left}(w)}^0 + \mathrm{e}_{\mathrm{right}(w)}^0, \mathrm{Env}_{\mathrm{right}(w)}^0) \tag{26}$$

Scheduling a regular or delayed task is equivalent to adding the task i^0 to Θ^0 or the task i^1 to Θ^1 by updating the node corresponding to the task according to (23) and (24). Afterwards, the values of the energy and envelope for the set of all tasks which are inserted in the tree so far can be recursively computed for the internal nodes of the tree. For at most k delayed tasks the contribution of at most j delayed tasks in the left and at most $k - j$ tasks in the right subtree emanating from w, $0 \leq j \leq k$, must be added up. Hence the energies of all tasks where at most k tasks are delayed, denoted e_w^k, and the sum of the earliest energy envelope of all tasks where at most k tasks are delayed, denoted Env_w^k, $0 \leq k \leq r$, are recursively computed as

$$\mathrm{e}_w^k = \max_{0 \leq j \leq k} \{\mathrm{e}_{\mathrm{left}(w)}^j + \mathrm{e}_{\mathrm{right}(w)}^{k-j}\} \tag{27}$$

$$\mathrm{Env}_w^k = \max_{0 \leq j \leq k} \{\mathrm{Env}_{\mathrm{left}(w)}^j + \mathrm{e}_{\mathrm{right}(w)}^{k-j}\} \cup \{\mathrm{Env}_{\mathrm{right}(w)}^k, \mathrm{Env}_w^{k-1}\} \tag{28}$$

In the case that the number of delayed task is greater than the number of available nodes in the subtree of the right side, it is sufficient to retrieve Env_w^{k-1}, i.e. the energy envelope for when at most $k - 1$ tasks are delayed. At the root of the Θ_L^r-tree, we obtain $\mathrm{Env}_{\mathrm{root}}^r = \mathrm{Env}^r(\Theta^0, \Theta^1)$. This quantity is essential to perform the overload checking.

Lemma 2 *The update of the Θ_L^r-tree runs in $\Theta(r^2 \log(n))$.*

Proof Upon the update of the values e_v^k and Env_v^k of a leaf as well as all nodes along the path connecting the leaf node to the root of the Θ_L^r-tree, there are r functions e_v^k to compute in constant time. There are also r functions Env_v^k, each one computed in $O(r)$ time, which deduces $\Theta(r^2 \log(n))$ computations. □

3.5 Robust overload checking algorithm

Let t_1 and t_2 be two arbitrary time points such that $t_1 < t_2$. The overload checking test ensures that the total energy of the tasks executing within the interval $[t_1, t_2]$ does not exceed the total energy available inside it. For the rule (15) it turns out that it suffices to check it for $t_1 = \mathrm{est}_i, t_2 = \mathrm{lct}_j^0$ for $i, j \in \mathcal{I}$. In the algorithm that we propose, t_2 could also be $t_2 = \mathrm{lct}_j^1$ for $j^1 \in \mathcal{I}^1$.

Algorithm 1: Overload checking($\mathcal{I}$, $\mathcal{C}$,r)

```
1 $\Theta^0 \leftarrow \emptyset$;
2 $\Theta^1 \leftarrow \emptyset$;
3 $T \leftarrow \{\mathrm{lct}_i^0 : i \in \mathcal{I}\} \cup \{\mathrm{lct}_i^1 : i \in \mathcal{I}\}$;
4 for $t \in T$ sorted in non-decreasing order do
5     $\Theta^0 \leftarrow \Theta^0 \cup \{i \in \mathcal{I} : \mathrm{lct}_i^0 = t\}$;
6     $\Theta^1 \leftarrow \Theta^1 \cup \{i \in \mathcal{I} : \mathrm{lct}_i^1 = t\}$;
7     if $\mathrm{Env}^r(\Theta^0, \Theta^1) > \mathcal{C} \cdot t$ then
8         fail;
```

In order to develop the overload checking rule in the robust context, et $T = \{\mathrm{lct}_i^0 : i \in \mathcal{I}\} \cup \{\mathrm{lct}_i^1 : i \in \mathcal{I}\}$ be the set of all latest completion time and delayed latest completion time points. The algorithm starts with an empty Θ_L^r-tree, i.e. $\Theta^0 = \Theta^1 = \emptyset$. That is, no tasks is initially scheduled. The idea is to process the time points $t \in T$ in non-decreasing order and schedule the regular tasks whose latest completion time is equal to t by adding them to Θ^0 or schedule the delayed tasks whose delayed latest completion time is equal to t by adding them to Θ^1. Such an addition changes the energy and envelopes for the leaves corresponding to the tasks in the tree as well as for all nodes on the branch connecting the leaves to the root. At each iteration, if a task i satisfies $\mathrm{lct}_i^0 = t$, then it is scheduled on the Θ_L^r-tree and i^0 is added to Θ^0 and if $\mathrm{lct}_i^1 = t$, then i is updated in the Θ_L^r-tree and i^1 is added to Θ^1. Once scheduling the tasks corresponding to t is over, the overload checking rule (21) must be assessed. Thanks to lemma 1, in order to assess (21) in the algorithm, it suffices to check $\mathrm{Env}^r(\mathrm{Lcut}^0(\mathrm{t}), \mathrm{Lcut}^1(\mathrm{t})) > \mathcal{C} \cdot \mathrm{t}$ for $t \in T$ which is being processed and $\mathrm{Env}^r(\mathrm{Lcut}^0(\mathrm{t}), \mathrm{Lcut}^1(\mathrm{t}))$ can be retrieved from the root of the Θ_L^r-tree that is developed so far. The algorithm 1 implements the overload checking algorithm in the robust context.

In order to elucidate the mechanism of the algorithm 1, we present few steps of implementing the algorithm for the example 1 and illustrate them in figure 1. Note that the nodes of the Θ_L^r-tree that are affected during the updates that occur at each step are discriminated in blue colours. For this example we have $\mathcal{I} = \{A, B, C, D, E\}$ and $T = \{14, 20, 21, 23, 24, 25, 26, 27\}$. After initializing an empty Θ_L^r-tree, in the first iteration the regular tasks i^0 for which $\mathrm{lct}_i^0 = 14$ and the delayed tasks i^1 for which $\mathrm{lct}_i^1 = 14$ are scheduled. C is the only tasks which qualifies in both conditions. Therefore, C^0 is added to Θ^0 and C^1 is added to Θ^1. The figure 1a depicts the status of the Θ_L^r-tree after this iteration. In the next iteration, A is the only eligible task to be schedule, but it is not allowed to be delayed, as $\mathrm{lct}_A^0 = 20$ and $\mathrm{lct}_A^1 = 21 > t = 20$. Therefore A^0 is added to Θ^0 but A^1 is not yet added to Θ^1. The figure 1b illustrates the status of the Θ_L^r-tree after this iteration. In the next iteration, corresponding to the figure 1c, A^1 and D^0 are scheduled, as $\mathrm{lct}_A^1 = \mathrm{lct}_D^0 = 21$. The figure 1d demonstrates the status of the Θ_L^r-tree at iteration $t = 25$. Before $t = 25$ is processed at this step $\Theta^0 = \{C^0, A^0, D^0, B^0\}$ and $\Theta^1 = \{C^1, A^1, D^1\}$. Since $\mathrm{lct}_B^1 = 25$, B^1 must be added to Θ^1. The insertion of B^1 into the Θ_L^r-tree causes the overload checking to fail, as $\mathrm{Env}^2(\Theta^0, \Theta^1) = \mathrm{Env}_{\mathrm{root}}^2 = 102 > 4 \cdot 25 = 100$.

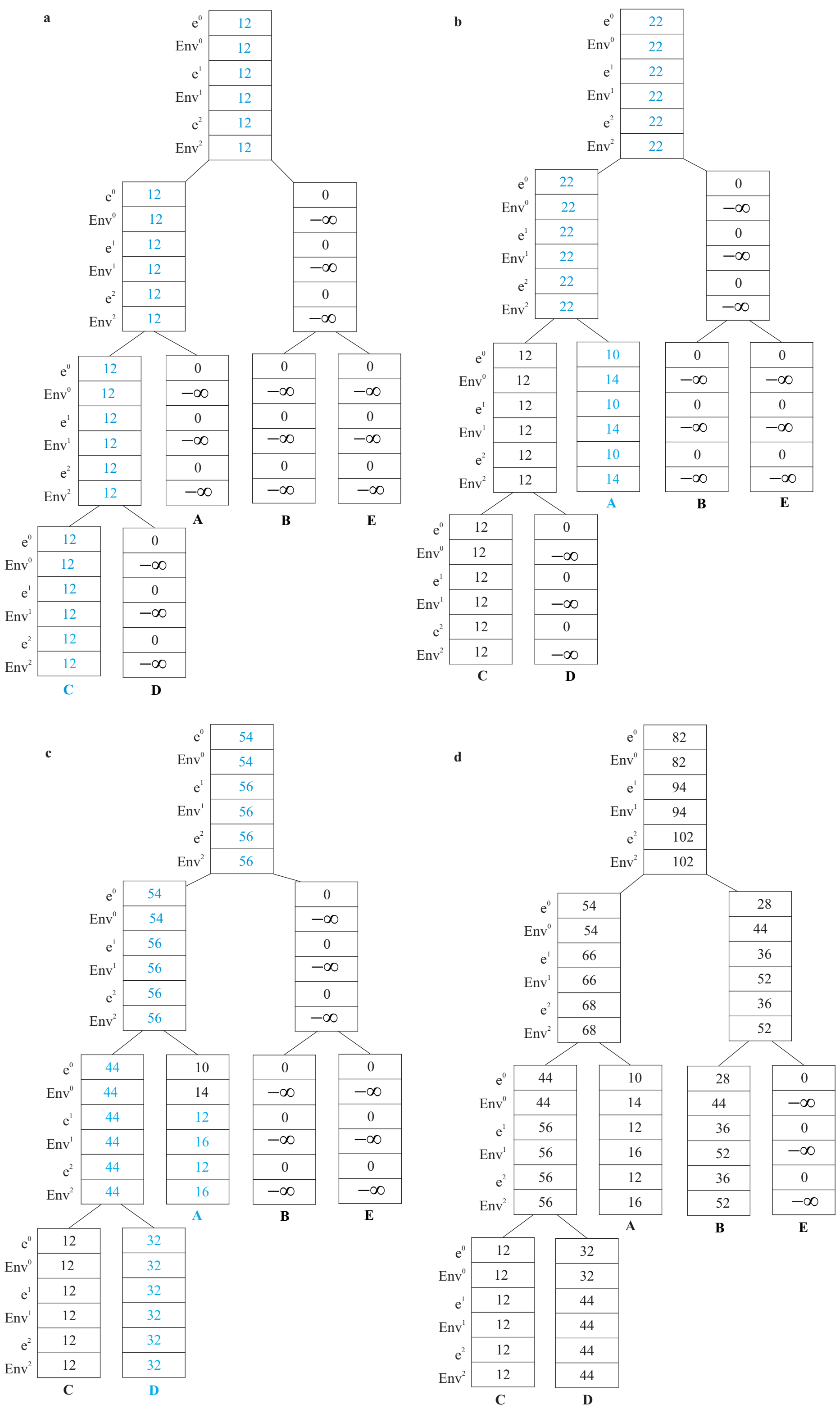


Fig. 1 In the picture (a), C^0 and C^1 are scheduled in the Θ_L^r-tree. The coloured nodes in blue represent the affected nodes during the update of the tree. In this case, $\Theta^0 = \{C^0\}$ and $\Theta^1 = \{C^1\}$. In the picture (b), A^0 is scheduled in the Θ_L^r-tree, but not A^1. In this case, $\Theta^0 = \{C^0, A^0\}$ and $\Theta^1 = \{C^1\}$. In the picture (c), A^1 and D are scheduled. In this case, $\Theta^0 = \{C^0, A^0, D^0\}$ and $\Theta^1 = \{C^1, A^1\}$. Picture (d) represents the status of the Θ_L^r-tree when processing $t = 25$. Since $\mathrm{Env}^2(\Theta^0, \Theta^1) = \mathrm{Env}^2_{\mathrm{root}} = 102 > 4 \cdot 25 = 100$ the overload checking triggers a failure.

Lemma 3 *The algorithm 1 runs in $\Theta(r^2 n \log(n))$.*

Proof According to lemma 2, the lines 5 and 6 of the algorithm 1 run in $\Theta(r^2 \log(n))$. Since every task is inserted exactly once in Θ^0 and once in Θ^1, overall the computational effort is $\Theta(r^2 n \log(n))$. □

4 Robust Edge-Finding

The objective of this section is to adapt the Edge-Finding algorithm introduced in [**?**] for the $\textsc{FlexC}([S_1, \dots, S_n], [p_1^0, \dots, p_n^0], [d_1, \dots, d_n], [\mathrm{c}_1, \dots, \mathrm{c}_n], \mathcal{C}, r)$ constraint. We present a generalization of the Edge-Finding detection rules (16) and (17) for the $\textsc{FlexC}([S_1, \dots, S_n], [p_1^0, \dots, p_n^0], [d_1, \dots, d_n], [\mathrm{c}_1, \dots, \mathrm{c}_n], \mathcal{C}, r)$ constraint. Prior to expound the mechanism of our algorithms, we should be mindful that in contrast to the regular scheduling problems, where the tasks are not assumed to delay, the robust framework is not symmetrical [**?**]. Indeed, a task can be delayed but it cannot be brought forward. Therefore, one can not simply apply the rule of filtering earliest starting times to the symmetrically negated problem in order to filter latest completion times. Therefore, we treat the filtering of lower bounds and upper bounds independently.

The structure of this section is as follows. Section 4.1 is concerned with filtering the earliest starting times. In the section 4.1.1 we establish the generic form of the robust Edge-Finding rule and accompany that with an example. In section 4.1.2 a new variant of robust earliest energy envelope for the conjunction of two disjoint subsets of tasks is defined. In section 4.1.3 we propose an extended data structure which enables us to handle the tasks when a certain number of them are delayed. Finally, in section 4.1.4 we take advantage of the introduced data structure to present the robust Edge-Finding algorithm. Moreover, we trace few steps of the algorithms for the example provided. Section 4.1 finishes with a discussion on the time complexity of the algorithm. Section 4.2 studies the filtering of latest completion times and follows the same structure as Section 4.1.

4.1 Filtering the earliest starting times

This section discusses the Edge-Finding rule, as well as the material required for detecting precedences among the tasks and adjusting earliest starting times.

4.1.1 Robust Edge-Finding rule for filtering the earliest starting times

Let $\Theta^0 \subseteq \mathcal{I}^0, \Theta^1 \subseteq \mathcal{I}^1$ and $i^b \in (\mathcal{I}^0 \cup \mathcal{I}^1) \setminus (\Theta^0 \cup \Theta^1)$ be such that $b \in \{0, 1\}$, $|\Theta^1| \leq r - b$ and $\mathcal{I}(\Theta^0) \cap \mathcal{I}(\Theta^1) = \emptyset$. Then the following states the Edge-Finding rule

$$\mathcal{C}(\max(\mathrm{lct}^0_{\mathcal{I}(\Theta^0)}, \mathrm{lct}^1_{\mathcal{I}(\Theta^1)}) - \mathrm{est}_{\mathcal{I}(\Theta^0 \cup \Theta^1 \cup \{i^b\})}) < \mathrm{e}_{\Theta^0} + \mathrm{e}_{\Theta^1} + \mathrm{e}_{i^b} \Rightarrow \Theta^0 \cup \Theta^1 \lessdot i^b \tag{29}$$

task	est	lct^0	p^0	d	c
A	1	13	4	1	6
B	6	23	9	1	7
C	16	36	8	10	2
D	14	28	12	1	6

Table 2 The set of tasks $\mathcal{I} = \{A, B, C, D\}$ to execute on a resource of capacity $\mathcal{C} = 7$ in Example 2.

Equation (29) is implied from the fact that if the execution of i^b along with a set $\Theta^0 \cup \Theta^1$ with at most $r - b$ delayed tasks causes a failure due to the overload checking, then i^b should complete after $\Theta^0 \cup \Theta^1$.

Example 2 Let $\mathcal{I} = \{A, B, C, D\}$ be the set of tasks in table 2 which must execute on a resource of capacity $\mathcal{C} = 7$ in the context where at most $r = 1$ tasks are allowed to delay. According to (29), the precedence $\{B^0, D^0\} \prec C^1$ is deduced for $\Theta^0 = \{B^0, D^0\}, \Theta^1 = \emptyset, i = C$ and $b = 1$, as 154 = 7(28 - 6) < 63 + 72 + 36 = 171 and the precedence $\{A^0, B^1\} \prec D^0$ holds for $\Theta^0 = \{A^0\}, \Theta^1 = \{B^1\}, i = D$ and $b = 0$, as 161 = 7(24 - 1) < 24 + 70 + 72 = 166.

4.1.2 Robust $\Lambda-$earliest energy envelope

The Edge-Finding can be implemented such that during the processing of the tasks, the regular or delayed tasks which make the precedences are maintained in a subset of tasks $\Lambda \subset \mathcal{I}$. Initially the tasks belong to Θ^0 and Θ^1 and the idea is to check whether adding one task from Λ^0 to Θ^0 or adding one task from Λ^1 to Θ^1 leads to $\text{Env}(\Theta^0, \Theta^1) > \mathcal{C} \cdot \max(\text{lct}^0_{\mathcal{I}(\Theta^0)}, \text{lct}^1_{\mathcal{I}(\Theta^1)})$. As soon as such a task is found in Λ, the established precedence is recorded and the task gets unscheduled from Λ. Assuming that Λ^0 and Λ^1 respectively contain the regular and delayed tasks from Λ, a variant of the earliest energy envelope of the tasks in $\Theta \cup \Lambda$, when one task from Λ is selected and at most r tasks are delayed, is defined as follows.

$$\text{Env}^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1) = \max(\max_{i^0 \in \Lambda^0} \text{Env}^r(\Theta^0 \cup \{i^0\}, \Theta^1), \max_{i^1 \in \Lambda^1} \text{Env}^r(\Theta^0, \Theta^1 \cup \{i^1\})) \tag{30}$$

$\text{Env}^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1)$ is the largest envelope that can be taken by taking a task from Λ^0 or Λ^1 and adding to Θ^0 and Θ^1. In the following, we present a data structure from which $\text{Env}^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1)$ can be retrieved efficiently.

4.1.3 $(\Theta - \Lambda)^r_L-$tree

Vilím [?] extends the $\Theta-$tree to a $(\Theta - \Lambda)-$tree. This tree is primarily different from $\Theta-$tree in that it keeps track of the tasks in Λ for which a precedence can exist. This is a feature that is not required to be addressed in the $\Theta-$tree. Similar to the $\Theta-$tree, the tasks are sorted by the earliest starting times as the leaves of $(\Theta - \Lambda)-$tree. Furthermore, the nodes of $(\Theta - \Lambda)-$tree maintain additional parameters for the energy and envelope of the tasks which belong to Λ. Analogous to the notion of Θ^r_L-tree, we define the $(\Theta - \Lambda)^r_L-$tree which is an extension of $(\Theta - \Lambda)-$tree. In addition to the parameters e^k_v and Env^k_v as in (23) and (24), this tree maintains the parameters $\Lambda-$energy, denoted $\mathrm{e}_v^{\Lambda k}$, and $\Lambda-$earliest energy envelope, denoted $\mathrm{Env}_v^{\Lambda k}$, associated to the tasks in Λ as follows.

$$\mathrm{e}_v^{\Lambda k} = \begin{cases} \mathrm{c}_v\, p_v^0 & \text{if } (v^0 \in \Lambda^0) \wedge (k = 0 \ \vee \ v^1 \notin \Lambda^1) \\ \mathrm{c}_v\, p_v^1 & \text{if } (v^1 \in \Lambda^1) \wedge (k > 0) \\ -\infty & \text{otherwise} \end{cases} \tag{31}$$

$$\mathrm{Env}_v^{\Lambda k} = \begin{cases} \mathcal{C}\,\mathrm{est}_v + \mathrm{e}_v^{\Lambda k} & \text{if } (v^0 \in \Lambda^0) \cup (v^1 \in \Lambda^1) \\ -\infty & \text{otherwise} \end{cases} \tag{32}$$

Similar to the Θ_L^r-tree, the superscript k stands for an upper bound on the number of delayed tasks in the leaf v. In order to make the computations of the $\Lambda-$energy and $\Lambda-$earliest energy envelope symmetrical over all nodes of the tree, we suppose that $0 \leq k \leq r$ and since $\mathrm{e}_v^{\Lambda k}$ and $\mathrm{Env}_v^{\Lambda k}$ are the $\Lambda-$energy and $\Lambda-$earliest energy envelope when at most k tasks are delayed, for $k \geq 2$ in the leaves of the $(\Theta - \Lambda)_L^r-$tree we necessarily have $\mathrm{e}_v^{\Lambda k} = \mathrm{e}_v^{\Lambda 1}$ and $\mathrm{Env}_v^{\Lambda k} = \mathrm{Env}_v^{\Lambda 1}$.

Let w be an internal node of the tree. $\mathrm{e}_w^{\Lambda k}$ and $\mathrm{Env}_w^{\Lambda k}$ are the maximum $\Lambda-$energy and the $\Lambda-$earliest energy envelope of the tasks in Θ whose leaves are descendant of w and to which one task from Λ is added. The task from Λ is also a descendant of w. It is proven [?] that

$$\mathrm{e}_w^{\Lambda 0} = \max(\mathrm{e}_{\mathrm{left}(w)}^0 + \mathrm{e}_{\mathrm{right}(w)}^{\Lambda 0}, \mathrm{e}_{\mathrm{left}(w)}^{\Lambda 0} + \mathrm{e}_{\mathrm{right}(w)}^0) \tag{33}$$

$$\mathrm{Env}_w^{\Lambda 0} = \max(\mathrm{Env}_{\mathrm{left}(w)}^{\Lambda 0} + \mathrm{e}_{\mathrm{right}(w)}^0, \mathrm{Env}_{\mathrm{right}(w)}^{\Lambda 0}, \mathrm{Env}_{\mathrm{left}(w)}^0 + \mathrm{e}_{\mathrm{right}(w)}^{\Lambda(0)}) \tag{34}$$

When scheduling, the regular tasks i^0 are added to Λ^0 and the delayed tasks i^1 are added to Λ^1. When unscheduling, the regular tasks i^0 are removed from Λ^0 and the delayed tasks i^1 are removed from Λ^1. Then, the nodes corresponding to the task are updated according to (31) and (32).

At most k tasks are delayed and in the computation of $\mathrm{e}_w^{\Lambda k}$ one task from Λ contributes. This task could be among at most j delayed tasks in the left subtree or at most $k - j$ tasks in the right subtree emanating from the inner node w, $0 \leq k \leq r, 0 \leq j \leq k$. According to (33) and (34)

$$\mathrm{e}_w^{\Lambda k} = \max_{0 \leq j \leq k} \{\mathrm{e}_{\mathrm{left}(w)}^j + \mathrm{e}_{\mathrm{right}(w)}^{\Lambda(k-j)}, \mathrm{e}_{\mathrm{left}(w)}^{\Lambda j} + \mathrm{e}_{\mathrm{right}(w)}^{k-j}\} \tag{35}$$

$$\mathrm{Env}_w^{\Lambda k} = \max_{0 \leq j \leq k} \{\mathrm{Env}_{\mathrm{left}(w)}^{\Lambda j} + \mathrm{e}_{\mathrm{right}(w)}^{k-j}, \mathrm{Env}_{\mathrm{left}(w)}^j + \mathrm{e}_{\mathrm{right}(w)}^{\Lambda(k-j)}\} \cup \{\mathrm{Env}_{\mathrm{right}(w)}^{\Lambda k}, \mathrm{Env}_w^{\Lambda(k-1)}\} \tag{36}$$

In the case that the number of delayed task is greater than the number of available nodes in the subtree of the right side, it is sufficient to retrieve $\mathrm{Env}_w^{\Lambda(k-1)}$ in (36), i.e. the lambda energy envelope when at most $k - 1$ tasks are delayed. Finally, if w is the root of $(\Theta - \Lambda)_L^r-$tree, the function $\mathrm{Env}^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1)$ can be computed by computing the value $\mathrm{Env}_{\mathrm{root}}^{\Lambda k}$ at the root node.

4.1.4 Robust Edge-Finding algorithm for filtering the earliest starting times

The implementation of Edge-Finding proceeds in two phases. Firstly, the existing precedences among the tasks are detected. Thereafter, the earliest starting time of the tasks subject to a precedence are adjusted.

Detection phase

Algorithm 2 adapts the detection phase of the Edge-Finding for the earliest starting times. This algorithm emulates the algorithm that Vilím [**?**] proposes. The algorithm starts with a full $(\Theta - \Lambda)^r_L$−tree, in which all the regular as well as the delayed tasks are scheduled in $\Theta = \Theta^0 \cup \Theta^1$ and $\Lambda = \Lambda^0 \cup \Lambda^1$ is empty. That is,

$$\Theta^0 = \mathcal{I}^0, \Theta^1 = \mathcal{I}^1, \Lambda^0 = \Lambda^1 = \emptyset$$

The algorithm iterates over the set of all latest completion times and delayed latest completion times $T = \{\mathrm{lct}^0_i : i \in \mathcal{I}\} \cup \{\mathrm{lct}^1_i : i \in \mathcal{I}\}$ in non-increasing order. First, the algorithm makes sure that the overload checking does not fail (line 10). If so, for every $t \in T$

$$\mathrm{Env}^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1) > \mathcal{C} \cdot t \tag{37}$$

is checked which captures the precedence. Thanks to the structure of the $(\Theta - \Lambda)^r_L$−tree, $\mathrm{Env}^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1)$ is retrieved from the root of $(\Theta - \Lambda)^r_L$−tree for each t. Line 13 retrieves the task subject to a precedence. It can be implemented in $O(\log(n))$

Algorithm 2: DetectionPhaseOfEdge-FindingForLowerBounds($\mathcal{I}$, $\mathcal{C}$, r)

```
1  $T \leftarrow \{\mathrm{lct}^0_l : l \in \mathcal{I}\} \cup \{\mathrm{lct}^1_l : l \in \mathcal{I}\}$;
2  for $i \in \{1, ..., n\}$ do
3      prec[$i$, 0] $\leftarrow -\infty$;
4      prec[$i$, 1]$\leftarrow -\infty$;
5  $\Theta^0 \leftarrow \mathcal{I}^0$;
6  $\Theta^1 \leftarrow \mathcal{I}^1$;
7  $\Lambda^0 \leftarrow \emptyset$;
8  $\Lambda^1 \leftarrow \emptyset$;
9  for $t \in T$ in non-increasing order do
10     if $\mathrm{Env}^r(\Theta^0, \Theta^1) > \mathcal{C} \cdot t$ then
11         fail;
12     while $\mathrm{Env}^{\Lambda r}_{\mathrm{root}} > \mathcal{C} \cdot t$ do
13         $i^b$ ←The task in $\Lambda^0 \cup \Lambda^1$ that maximizes $\mathrm{Env}^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1)$ ;
14         prec[$i, b$] $\leftarrow t$ ;
15         $\Lambda^b \leftarrow \Lambda^b \setminus \{i^b\}$;
16     $\Delta^1 = \{i^1 \in \mathcal{I}^1 \mid \mathrm{lct}^1_i = t\}$;
17     $\Delta^0 = \{i^0 \in \mathcal{I}^0 \mid \mathrm{lct}^0_i = t\}$;
18     $\Lambda^1 \leftarrow \Lambda^1 \cup \Delta^1$;
19     $\Theta^1 \leftarrow \Theta^1 \setminus \Delta^1$;
20     $\Lambda^0 \leftarrow \Lambda^0 \cup \Delta^0$;
21     $\Theta^0 \leftarrow \Theta^0 \setminus \Delta^0$;
```

time by traversing down the tree. The algorithm proceeds by traversing down the $(\Theta - \Lambda)^r_L-$tree from the root. At each inner node w in such a traversal, the algorithm determines which one of the cases in (36) satisfy for $\mathrm{Env}^{\Lambda k}_w$. So long as this task is taken from $\mathrm{Env}^{\Lambda k}_f$, where f is a child of w, the algorithm continues down. As soon as the task is taken from an $\mathrm{e}^{\Lambda k}_f$, the algorithm switches to check the cases of (35). The task subject to a precedence is a task $i^b \in \Lambda^b, b \in \{0,1\}$, and only one task in $\Lambda^0 \cup \Lambda^1$ is used to compute $\mathrm{Env}^{\Lambda k}_w$. The candidate task with such a property is called the *responsible task*. The responsible task, which is located on a leaf of the $(\Theta - \Lambda)^r_L-$tree, causes (30) to be maximized, hence making a precedence. Contingent upon the delay status of i, we encode the precedence that i creates in a two dimensional matrix by 0 and 1 columns, indicating whether the task is regular or delayed (line 14). The precedences detected during the detection phase are encoded in the *prec* matrix. For $b \in \{0,1\}$, $\mathrm{prec}[i,b] = t$ means that i^b is preceded by the subsets of tasks from $\Theta^0 \cup \Theta^1$ with at most $r - b$ delayed tasks which terminate no later than t and cause (30) to be maximized. Once the precedence is recorded one line 14, the algorithm unschedules i^b from Λ^b. After the execution of the loop at line 12, all the tasks whose delayed latest completion time equals t are unscheduled from Θ^1 and they are rather scheduled in Λ^1. Furthermore, all the tasks whose latest completion time equals t, are removed from Θ^0 and Θ^1 and scheduled in Λ^0 and Λ^1.

In order to elucidate the mechanism of the algorithm 2, in figure 2 we present the first few steps of the detection phase for the tasks of example 2. For this example, $T = \{46, 36, 29, 28, 24, 23, 14, 13\}$. Figure 2a illustrates the initialization step of the algorithm, where the $(\Theta - \Lambda)^r_L-$tree is full. That is, all the regular as well as delayed tasks are scheduled and no task is a candidate yet to create a precedence.

In the first iteration $t = 46$ is processed, which corresponds to $\mathrm{lct}^1_C = t$. C^1 gets unscheduled from Θ^1 and rather scheduled in Λ^1, to be flagged as a delayed tasks which could create a precedence later. The updated status of the node corresponding to C^1 is depicted in figure 2b. In the second iteration, $t = 36$ is processed, which corresponds to $\mathrm{lct}^0_C = t$. At this point, $\mathrm{Env}^{\Lambda r}_{\mathrm{root}} = 213 \not> \mathcal{C} \cdot t$, which implies that (30) is not great enough to detect a precedence. Therefore, the loop dedicated to detect an existing precedence fails to execute and C^0 is removed from Θ^0 and fully scheduled in Λ. Figure 2c corresponds to this case after updating the entire tree. The next iteration processes $t = \mathrm{lct}^1_D = 29$. Since $\mathrm{Env}^{\Lambda r}_{\mathrm{root}}$ =213 > $\mathcal{C} \cdot t$, the while loop executes. A traversal down the tree to find the responsible task locates C^1. Once the precedence $\mathrm{prec}[C^1, 1] = 29$ is recorded, C^1 gets unscheduled from Λ^1, as illustrated in figure 2d. For this iteration there is no more precedences to detect. Thus, the while loop terminates and D^1 for which $\mathrm{lct}^1_D = 29$ gets unscheduled from Θ^1 and scheduled in Λ^1 as depicted in figure 2e. In further executions of the algorithm, the precedence $\{A^0, B^1\} \prec D^0$ will be detected, as well. Table 3 indicates the *prec* matrix as the result of the algorithm 2.

Adjustment phase

The adjustment of the earliest starting times is done by iterating over the detected precedences which are recorded in the prec matrix. Let i^b be the tasks for which a precedence was detected with (29) and and $\mathrm{prec}[i,b] = t$. For $\Omega^0 \subseteq \Theta^0, \Omega^1 \subseteq$

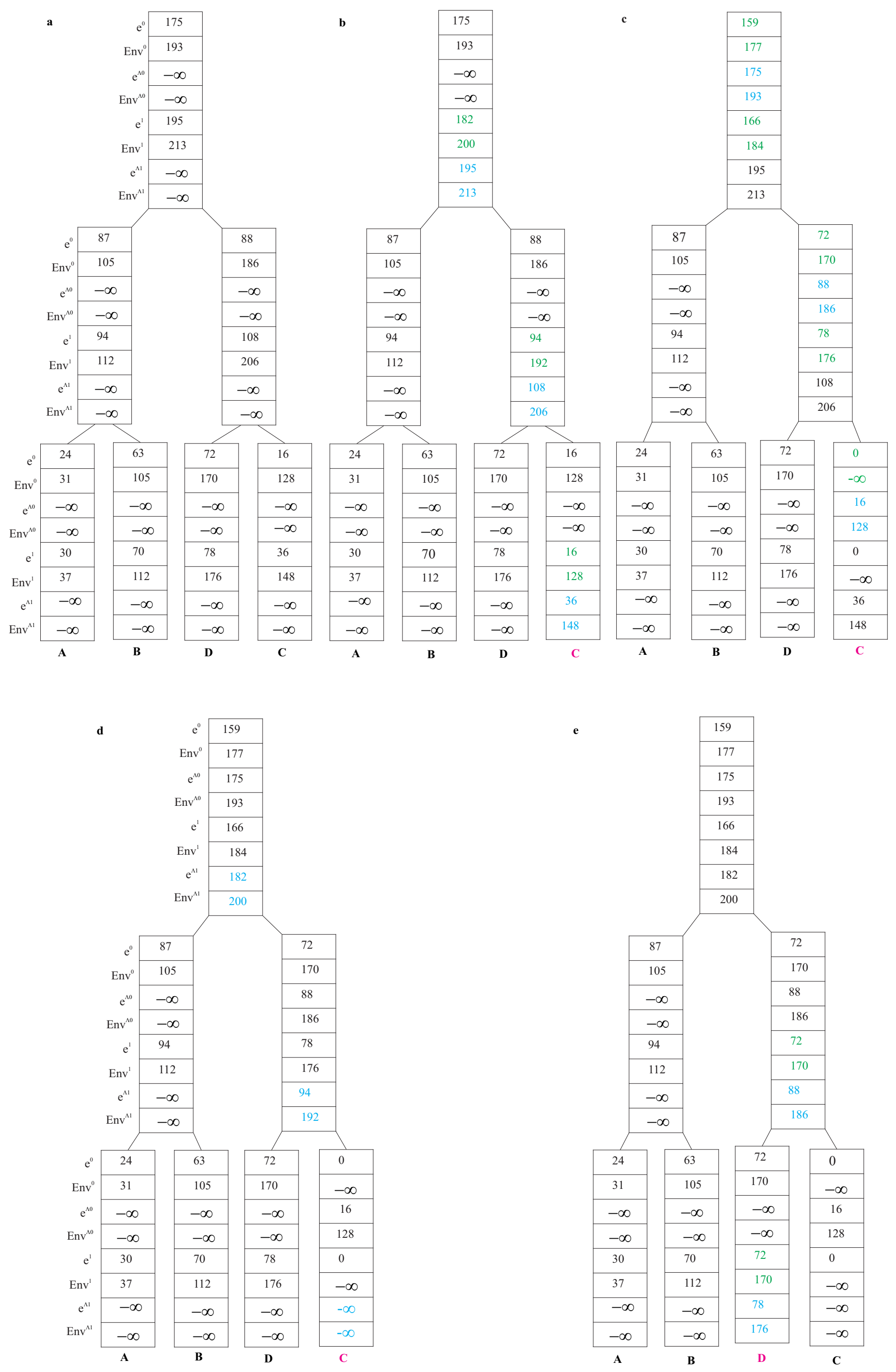


Fig. 2 In the picture (I), the algorithm starts with a full $(\Theta - \Lambda)^r_L$−tree. In the picture 2b, C^1 gets unscheduled from Θ^1 and scheduled in Λ^1. The modifications to Θ and Λ sets are respectively coloured in green and blue. In the picture 2c, C^0 is unscheduled from Θ^0 and scheduled in Λ^0. In the picture 2d after C^1 is found as the responsible tasks, it gets unscheduled from Λ^1. In the picture 2e, D^1 gets unscheduled from Θ^1 and scheduled in Λ^1.

b / Task	0	1
A	$-\infty$	$-\infty$
B	14	14
C	28	29
D	24	24

Table 3 The *prec* matrix which is obtained after the execution of the algorithm 2 on the instance of example 2.

$\Theta^1, \emptyset \neq \Omega^0 \cup \Omega^1$ such that $|\Omega^1| \leq r - b$, we define a variation of (18) in the robust context as:

$$\operatorname{rest}(\Omega^0, \Omega^1, \mathrm{c}_i) = \mathrm{e}_{\Omega^0 \cup \Omega^1} - (\mathcal{C} - \mathrm{c}_i)(\max(\operatorname{lct}^0_{\mathcal{I}(\Omega^0)}, \operatorname{lct}^1_{\mathcal{I}(\Omega^1)}) - \operatorname{est}_{\Omega^0 \cup \Omega^1}) \quad (38)$$

The following formula adjusts the earliest starting time of i.

$$\operatorname{est}_i \leftarrow \max(\operatorname{est}_i, \max_{\substack{\Omega^0 \subseteq \operatorname{Lcut}^0(t) \setminus \{i^0\} \cap \{j^0 \in \mathcal{I}^0 : \operatorname{lct}^0_j \leq t < \operatorname{lct}^1_j\} \\ \Omega^1 \subseteq \operatorname{Lcut}^1(t) \setminus \{i^1\} \\ \emptyset \neq \Omega^0 \cup \Omega^1 \\ |\Omega^1| \leq r - b \\ \operatorname{rest}(\Omega^0, \Omega^1, \mathrm{c}_i) > 0}} \{\operatorname{est}_{\Omega^0 \cup \Omega^1} + \left\lceil \frac{\operatorname{rest}(\Omega^0, \Omega^1, \mathrm{c}_i)}{\mathrm{c}_i} \right\rceil\}) \quad (39)$$

From the condition $\operatorname{rest}(\Omega^0, \Omega^1, \mathrm{c}_i) > 0$ we obtain

$$(\mathcal{C} - \mathrm{c}_i)(\operatorname{est}_{\Omega^0 \cup \Omega^1}) + \mathrm{e}_{\Omega^0 \cup \Omega^1} > (\mathcal{C} - \mathrm{c}_i)(\max(\operatorname{lct}^0_{\mathcal{I}(\Omega^0)}, \operatorname{lct}^1_{\mathcal{I}(\Omega^1)})) \quad (40)$$

The left side of (40) describes an equivalent of $\operatorname{Env}^r(\Theta^0, \Theta^1)$, defined by (22), on a resource with capacity $\mathcal{C} - \mathrm{c}_i$. This motivates the idea of defining a variant of the earliest energy envelope, defined as (4), with respect to each $c \in F$ [**?**] as

$$\operatorname{Env}^{\mathrm{c}}_{\Theta} = \max_{\Omega \subseteq \Theta}((\mathcal{C} - \mathrm{c}) \operatorname{est}_{\Omega} + \mathrm{e}_{\Omega}) \quad (41)$$

and a variant of (22) as

$$\operatorname{Env}^{\mathrm{c}\, r}(\Theta^0, \Theta^1) = \max_{\substack{\Omega^0 \subseteq \Theta^0 \\ \Omega^1 \subseteq \Theta^1 \\ |\Omega^1| \leq r \\ \mathcal{I}(\Omega^0) \cap \mathcal{I}(\Omega^1) = \emptyset}} ((\mathcal{C} - \mathrm{c}) \operatorname{est}_{\mathcal{I}(\Omega^0 \cup \Omega^1)} + \mathrm{e}_{\Omega^0} + \mathrm{e}_{\Omega^1}) \quad (42)$$

The algorithm 5 is our adaption of the adjustment phase of the Edge-Finding for this purpose. For the rest of this section, we explain how this algorithm emulates the algorithm that Vilím proposes [**?**].

Let F be the set of all distinct capacities. In [**?**] the adjustment is done by processing all $\mathrm{c} \in F$ in an outer loop and initializing a Θ^{c}-tree for each $\mathrm{c} \in F$ in an inner loop. Θ^{c}-tree is an extension of Θ-tree in that in addition to maintaining the parameters energy and earliest energy envelope, every inner node holds the additional

Algorithm 3: Maxest(tree,bound,c,$\mathcal{C}$,k,o)

```
1  $v \leftarrow \text{root};$
2  $\text{e}[0...k] \leftarrow \mathbf{0}$ ;
3  $\text{maxEnv}^c \leftarrow (\mathcal{C} - \text{c}) \cdot \text{bound};$
4  $k \leftarrow k - o;$
5  while $v$ is not a leaf do
6      branchRight ← false;
7      for $j = 0, ..., k$ do
8          if $\text{Env}^{c,j}_{\text{right(v)}} + \text{e}[k - j] > \text{maxEnv}^c$ then
9              $v \leftarrow \text{right}(v);$
10             branchRight ← true;
11             break;
12     if not branchRight then
13         $v \leftarrow \text{left}(\text{v});$
14         for $j = 0, ..., k$ do
15             $\text{e}'[j] \leftarrow \max_{0 \le i \le j}(\text{e}[i] + \text{e}^{j-i}_{\text{right(v)}}, \text{e}'[j-1])$
16         $\text{e} \leftarrow \text{e}';$
17 return $v$;
```

parameter (41). Analogous to the parameters (3) and (4) in the $\Theta-$tree, for a leaf v of the $\Theta^{\text{c}}-$tree, which corresponds to a task $i \in \mathcal{I}$, (41) is computed for $\Theta = \{i\}$ and for an inner node w of the $\Theta^{\text{c}}-$tree, (41) is computed for the set of all tasks which are

Algorithm 4: EnvelopeForLowerBound(v,tree,k)

```
1  $\text{e}_\alpha \leftarrow [\text{e}^0_v, \text{e}^1_v, 0, ..., 0]$; // $\text{e}_\alpha$ includes $k$ - 1 entries 0
2  $\text{e}_\beta \leftarrow \mathbf{0}$ ;
3  $\text{Env}_\alpha \leftarrow [\text{Env}^0_v, \text{Env}^1_v, -\infty, ..., -\infty]$; // $\text{Env}_\alpha$ includes $k$ - 1 entries $-\infty$
4  while $v$ is not the root do
5      if $v$ is a left child then
6          $\text{e}'_\beta \leftarrow \mathbf{0}$ ;
7          for $j = 0$ to $k$ do
8              $\text{e}'_\beta[j] \leftarrow \max_{0 \le i \le j}(\text{e}_\beta[i] + \text{e}^{j-i}_{\text{sibling}(v)}, \text{e}'_\beta[j-1])$
9          $\text{e}_\beta \leftarrow \text{e}'_\beta;$
10     else
11         $\text{Env}'_\alpha \leftarrow [-\infty, \ldots, -\infty];$
12         $\text{e}'_\alpha \leftarrow \mathbf{0}$ ;
13         for $j = 0, ..., k$ do
14             $\text{Env}'_\alpha[j] \leftarrow$
                 $\max(\max_{0 \le i \le j}(\text{Env}^i_{\text{sibling}(v)} + \text{e}_\alpha[j-i]), \text{Env}_\alpha[j], \text{Env}'_\alpha[j-1]);$
15             $\text{e}'_\alpha[j] \leftarrow \max(\max_{0 \le i \le j}(\text{e}^i_{\text{sibling}(v)} + \text{e}_\alpha[j-i]), \text{e}'_\alpha[j-1]);$
16     $\text{Env}_\alpha \leftarrow \text{Env}'_\alpha;$
17     $\text{e}_\alpha \leftarrow \text{e}'_\alpha;$
18     $v \leftarrow \text{parent}(v);$
19 return $\max_{0 \le i \le k}(\text{e}_\beta[i] + \text{Env}_\alpha[k-i]);$
```

Algorithm 5: AdjustmentOfLowerBounds(prec, r, $\mathcal{C}$)

```
1  $F \leftarrow$ The capacities of the tasks for which a precedence was detected;
2  for c $\in F$ do
3      $\Theta^0 \leftarrow \emptyset$;
4      $\Theta^1 \leftarrow \emptyset$;
5      upd $\leftarrow [-\infty, -\infty]$;
6      $t' \leftarrow 0$;
7      for $i^b \in \{j^b \mid prec[j,b] > -\infty \wedge \mathrm{c}_i = \mathrm{c}\}$ in non-deccreasing order of prec[i, b] do
8          $t \leftarrow \mathrm{prec}[i,b]$;
9          $\Theta^1 \leftarrow \Theta^1 \cup (\mathrm{Lcut}^1(t) \setminus \mathrm{Lcut}^1(t')) \setminus \{i^1\}$ ;
10         $\Theta^0 \leftarrow \Theta^0 \cup \{j^0 \in \mathcal{I}^0 : \mathrm{lct}_j^0 \leq t < \mathrm{lct}_j^1\} \setminus \{j^0 \in \mathcal{I}^0 : \mathrm{lct}_j^0 \leq t' < \mathrm{lct}_j^1\}) \setminus \{i^0\}$ ;
11         $m \leftarrow \max(\{\mathrm{lct}_j^0 \mid j^0 \in \Theta^0\} \cup \{\mathrm{lct}_j^1 \mid j^1 \in \Theta^1\})$ ;
12         $v \leftarrow \mathrm{Maxest}(\Theta, m, c, \mathcal{C}, r, b)$;
13         Env $\leftarrow \mathrm{EnvelopeForLowerBound}(v, \Theta, r-b)$ ;
14         diff $\leftarrow \lceil(\mathrm{Env} - (\mathcal{C} - \mathrm{c})m/\,\mathrm{c}\rceil$ ;
15         upd[b] $\leftarrow$ max(upd[b],diff) ;
16         $\mathrm{est}_i \leftarrow \max(\mathrm{est}_i$, upd[b] );
17         if $\mathrm{lct}_i^0 \leq t \wedge \mathrm{lct}_i^1 \leq t$ then $\Theta^1 \leftarrow \Theta^1 \cup \{i^1\}$;
18         else if $\mathrm{lct}_i^0 \leq t$ then $\Theta^0 \leftarrow \Theta^0 \cup \{i^0\}$ ;
19         $t' \leftarrow t$;
```

included in the subtree rooted at w. This computation is done recursively as below.

$$\mathrm{Env}_w^{\mathrm{c}} = \max(\mathrm{Env}_{\mathrm{left}(w)}^{\mathrm{c}} + \mathrm{e}_{\mathrm{right}(w)}^{0}, \mathrm{Env}_{\mathrm{right}(w)}^{\mathrm{c}}) \tag{43}$$

Note that in [?] all distinct capacities $\mathrm{c} \in F$ are considered individually, no matter a precedence was detected per capacity or not. For the sake of efficiency, we rather consider F as the set of capacities of the tasks for which a precedence was detected. For each $\mathrm{c} \in F$, we initialize an empty $\Theta_L^{\mathrm{c}\,r}-$tree, which is an extension of $\Theta_L^{r}-$tree. With regard to the material presented in section 3.4, a leaf v of the $\Theta_L^{\mathrm{c}\,r}-$tree in addition to (23) and (24) holds

$$\mathrm{Env}_v^{\mathrm{c}\,k} = \begin{cases} (\mathcal{C} - \mathrm{c})\,\mathrm{est}_v + \mathrm{e}_v^k & \text{if } (v^0 \in \Theta^0) \vee (v^1 \in \Theta^1) \\ -\infty & \text{otherwise} \end{cases} \tag{44}$$

and $\mathrm{Env}_w^{\mathrm{c}\,k}$ for an internal node w is computed as

$$\mathrm{Env}_w^{\mathrm{c}\,k} = \max_{0 \leq j \leq k} \{\mathrm{Env}_{\mathrm{left}(w)}^{\mathrm{c}\,j} + \mathrm{e}_{\mathrm{right}(w)}^{k-j}\} \cup \{\mathrm{Env}_{\mathrm{right}(w)}^{\mathrm{c}\,k}, \mathrm{Env}_w^{\mathrm{c}(k-1)})\} \tag{45}$$

Θ^{c}-tree develops by processing all lct_j^0 for $j \in \mathcal{I}$ in non-decreasing order and scheduling j by adding it to Θ. Rather than iterating through all lct_j^0 and lct_j^1 for $j \in \mathcal{I}$, we develop the $\Theta_L^{\mathrm{c}\,r}-$tree by iterating over the tasks i^b in non-decreasing order of $\mathrm{prec}[i,b]$. The adjustment should be done only if $\mathrm{c}_i = \mathrm{c}$. If so, at iterating each $\mathrm{prec}[i,b] = t$, all the regular or delayed tasks which cannot complete after t are scheduled in the $\Theta_L^{\mathrm{c}\,r}-$tree by adding to Θ^0 or Θ^1.

Let $\Omega^0 \cup \Omega^1$ be the candidate subset which can give the strongest adjustment in (39). Once all the tasks which are eligible to be in $\Omega^0 \cup \Omega^1$ are scheduled in the tree, it is time to identify Ω^0 and Ω^1 and compute the value of the second component of the right side of (39). Emulating [**?**], we proceed to compute Ω^0 and Ω^1 in two steps.

The first step is to compute the maximum earliest starting time (or *maxest*) for $\Omega^0 \cup \Omega^1$. There might be multiple sets $\Omega^{0\prime}$ and $\Omega^{1\prime}$ that satisfy (39) and the goal is to compute the largest $\mathrm{est}_{\Omega^{0\prime}\cup\Omega^{1\prime}}$ for the appropriate $\Omega^{0\prime}$ and $\Omega^{1\prime}$. The new variant of the earliest energy envelope, defined in (41), helps to locate the node responsible for maxest. In the algorithms 3, we consider this idea for identifying maxest for $\Omega^0 \cup \Omega^1$ in (39). Note that line 11 of algorithm 5 retrieves the largest completion time for the set of all tasks that are scheduled so far, whether they are regular or delayed. This value, named *bound*, which is given as a parameter to the algorithm 3, is important for checking the right side of (40) in this algorithm.

The second step is to compute the envelope of the tasks which is done in the algorithm 4 . This is done by dividing the tasks in two groups. The tasks which start before or at maxest and the rest of the tasks which start after maxest. The tasks which qualify for the former case belong to a set $\alpha(j, \mathrm{c})$ and the tasks which qualify for the latter case belong to a set $\beta(j, c)$. From the sets $\alpha(j, \mathrm{c})$ and $\beta(j, c)$, the earliest energy envelop in the $\Theta_L^{\mathrm{c}\,r}-$tree is computed with

$$\mathrm{Env}(j, c) = \mathrm{e}_\beta + \mathrm{Env}_{\alpha(j,\mathrm{c})} \tag{46}$$

where e_β and Env_α are computed in a bottom-up manner, starting from the located task responsible for maxest, and by taking into account that at most r tasks are delayed. Once the envelope is computed, the value which makes the strongest update is adjusted by taking the maximum between the envelope computed in the current iteration and the preceding iterations (line 15). Ultimately, if the task can not finish after t, it gets scheduled itself.

Here is the adjustment of the precedence $\{B^0, D^0\} \prec C^1$ for the task C from the example 2. Figure 3 illustrates the $\Theta_L^{\mathrm{c}\,r}-$tree for the state where est_C is adjusted. The adjustment is done when processing $\mathrm{c} = 2$. At this state, $\Theta^0 = \{A^0, B^0, D^0\}$ and $\Theta^1 = \{A^1, B^1\}$. The set of scheduled tasks has, for largest completion completion time, $m = 28$ in line 11 of the algorithm. The blue arrows in figure 3 show the path which is traversed by the algorithm 3. This algorithm locates the yellow node which corresponds to the task D for maxest. The algorithm 4 starts from the yellow node and the green arrows show the path which is actually the reversed blue path traversed by this algorithm to the root. This algorithm computes $\mathrm{Env} = 184$ for line 13 and in line 14 the difference is computed diff = $\lceil 184 - (7-2)28/2 \rceil = 22$. Hence, est_C gets filtered to $\mathrm{est}_C = 22$. Proceeding the adjustment phase for the precedence $\{A^0, B^1\} \prec D^0$ in a similar fashion yields $\mathrm{est}_D = 15$.

Lemma 4 *The time complexity of the robust Edge-Finding is $O(r^2 kn \log(n))$.*

Proof Unscheduling a task from Θ^0 or Θ^1 or scheduling in Λ^0 or Λ^1 requires to update the values e_i and Env_i, $\mathrm{e}_v^{\Lambda j}$ and $\mathrm{Env}_v^{\Lambda j}$ of the leaf of the tree as well as all nodes up to the root of the tree. Therefore, the lines 18, 19, 20 and 21 of the algorithm 2 run in $O(r^2 \log(n))$. This complexity is maintained for finding the responsible task, finding the maxest as well as computing the envelope, since for such operations the tree is traversed from the root to the leaves or conversely. The scheduling and unscheduling

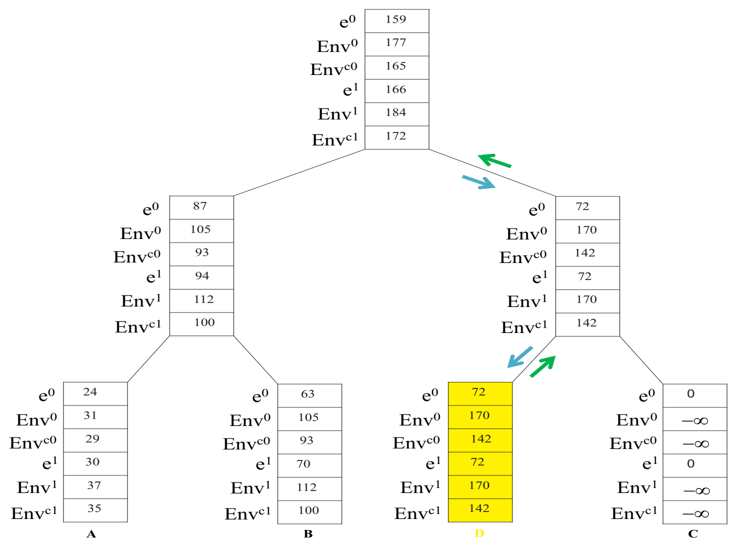


Fig. 3 The state of $\Theta^{c\,r}_{L}-$tree in the iteration that est_C gets adjusted. The blue arrows show the path which is traversed by the algorithm 3. This algorithm locates the yellow node. The algorithm 4 starts from the yellow node and the green arrows show the path traversed by this algorithm to the root.

tasks at lines 9 and 10 of the algorithm 5 occur at most n times, each time implying in $O(r^2 \log(n))$ computations. Moreover, since each task is unscheduled once from Θ^0 and once from Θ^1 or scheduled once in Λ^0 and once in Λ^1, considering that there are k distinct capacities, the overall time complexity is $O(r^2 kn \log(n))$. □

4.2 Filtering the latest completion times

This section discusses the Edge-Finding rule, as well as the material required for detecting precedences among the tasks and adjusting latest completion times.

4.2.1 Robust Edge-Finding rules for filtering the latest completion times

Let $\Theta^0 \subseteq \mathcal{I}^0, \Theta^1 \subseteq \mathcal{I}^1$ and $i^0 \in \mathcal{I}^0 \setminus \Theta^0$ be such that $\mathcal{I}(\Theta^0) \cap \mathcal{I}(\Theta^1) = \emptyset$. The following states the first Edge-Finding rule

$$\mathcal{C}(\max(\mathrm{lct}^0_{\mathcal{I}(\Theta^0)\cup\{i\}}, \mathrm{lct}^1_{\mathcal{I}(\Theta^1)}) - \mathrm{est}_{\mathcal{I}(\Theta^0\cup\Theta^1)}) < \mathrm{e}_{\Theta^0} + \mathrm{e}_{\Theta^1} + \mathrm{e}_{i^0} \Rightarrow i^0 \prec \Theta^0 \cup \Theta^1 \tag{47}$$

If $i^1 \in \mathcal{I}^1 \setminus \Theta^1$ be such that $\mathcal{I}(\Theta^0) \cap \mathcal{I}(\Theta^1) = \emptyset$, then the following states the second Edge-Finding rule

$$\mathcal{C}(\max(\mathrm{lct}^0_{\mathcal{I}(\Theta^0)}, \mathrm{lct}^1_{\mathcal{I}(\Theta^1)\cup\{i\}}) - \mathrm{est}_{\mathcal{I}(\Theta^0\cup\Theta^1)}) < \mathrm{e}_{\Theta^0} + \mathrm{e}_{\Theta^1} + \mathrm{e}_{i^1} \Rightarrow i^1 \prec \Theta^0 \cup \Theta^1 \tag{48}$$

task	est	lct^0	p^0	d	c
A	1	13	4	1	6
B	6	23	9	1	7
C	22	36	8	10	2
D	15	28	12	1	6

Table 4 Tasks of Example 2 with updated earliest starting times

Example 3 After adjusting est_C and est_D due to the precedences detected for the tasks C and D in example 2, the tasks are updated as in Table 4. This update causes two new precedences to be detected by filtering the latest completion times. Assuming $\Theta^0 = \{B^0\}, \Theta^1 = \{D^1\}, i = A, b = 0$, the precedence $A^0 \prec \{D^1, B^0\}$ holds as 161 = 7(29 - 6) < 24 + 63 + 78 = 165 and assuming $\Theta^0 = \{B^0\}, i = D, b = 0$ the precedence $B^0 \prec D^0$ holds as 91 = 7(28 - 15) < 63 +72 = 135.

4.2.2 Robust latest energy envelope

One can generalize the notion of the latest energy envelope, defined with (5), for the case that the tasks can be delayed. Rather than computing the latest energy envelope of a set Θ as in (5), we compute the latest energy envelope of two sets $\Theta^0 \subseteq \mathcal{I}^0$ and $\Theta^1 \subseteq \mathcal{I}^1$. The tasks in Θ^0 can be regular while the tasks in Θ^1 can be delayed. The tasks that belong to both sets can either be regular or delayed but not both. The latest energy envelope for the sets Θ^0 and Θ^1 in such a case is defined

$$\mathrm{Env}'^r(\Theta^0, \Theta^1) = \min_{\substack{\Omega^0 \subseteq \Theta^0 \\ \Omega^1 \subseteq \Theta^1 \\ |\Omega^1| \le r \\ \mathcal{I}(\Omega^0) \cap \mathcal{I}(\Omega^1) = \emptyset}} \left(\mathcal{C} \max(\mathrm{lct}^0_{\mathcal{I}(\Omega^0)}, \mathrm{lct}^1_{\mathcal{I}(\Omega^1)}) - \mathrm{e}_{\Omega^0} - \mathrm{e}_{\Omega^1}\right) \tag{49}$$

4.2.3 Robust $\Lambda-$latest energy envelope

Similar to filtering the earliest starting times, the Edge-Finding for filtering the latest completion times can also be implemented such that during the processing of the tasks, the regular or delayed tasks which make the precedences are maintained in a subset of tasks $\Lambda = \Lambda^0 \cup \Lambda^1 \subset \mathcal{I}$. Initially the tasks belong to Θ and the idea is to check whether adding one task from Λ^0 to Θ or adding one task from Λ^1 to Θ leads to $\mathrm{Env}'(\Theta^0, \Theta^1) < \mathcal{C} \cdot \mathrm{est}_{\mathcal{I}(\Theta^0 \cup \Theta^1)}$ for $\Theta^0 = \{i^0 \mid i \in \Theta\}$ and $\Theta^1 = \{i^1 \mid i \in \Theta\}$. As soon as such a task is found in Λ, the established precedence is recorded and the task gets unscheduled from Λ. Assuming that Λ^0 and Λ^1 respectively contain the regular and delayed tasks from Λ, a variant of the latest energy envelope of the tasks in $\Theta \cup \Lambda$, when one task from Λ is selected and at most r tasks are delayed, is defined as follows.

$$\mathrm{Env}'^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1) = \min\left(\min_{i^0 \in \Lambda^0} \mathrm{Env}'^r(\Theta^0 \cup \{i^0\}, \Theta^1), \min_{i^1 \in \Lambda^1} \mathrm{Env}'^r(\Theta^0, \Theta^1 \cup \{i^1\})\right) \tag{50}$$

$\mathrm{Env}'^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1)$ is the smallest envelope that can be taken by taking a responsible task from Λ^0 or Λ^1 and adding to Θ^0 and Θ^1. In the following, we present a data structure from which $\mathrm{Env}'^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1)$ can be retrieved efficiently.

4.2.4 $(\Theta - \Lambda)^r_U$*−tree*

Analogous to the notion of $(\Theta - \Lambda)^r_L$−tree, we define the $(\Theta - \Lambda)^r_U$−tree in order to develop the detection algorithm for filtering latest completion times. However, compared with the the preceding section, there is a major discrepancy in the way we construct the $(\Theta - \Lambda)^r_U$−tree. We associate each task i to two leaves. The regular task i^0 is associated to a leaf as usual but the delayed task i^1 is associated to another leaf that takes into account only the delayed part of the task. One can interpret it as a task with processing time d_i. Moreover, the values computed in each node of the tree are a function of three sets: $\Theta \subseteq \mathcal{I}$, $\Lambda^0 \subset \mathcal{I}^0$, and $\Lambda^1 \subset \mathcal{I}^1$. For instance, with such an interpretation, in the example 3, Λ^0 has $p^0_A \cdot \mathrm{c}_A = 4 \cdot 6 = 24$ units of energy, while i^1 has $d_A \cdot \mathrm{c}_A = 1 \cdot 6 = 6$ units of energy.

Let $T = \{\mathrm{lct}^0_i : i \in \mathcal{I}\} \cup \{\mathrm{lct}^1_i : i \in \mathcal{I}\}$ be the set of all latest completion times and delayed latest completion times sorted in ascending order. We construct the $(\Theta - \Lambda)^r_U$−tree with $2n$ leaves. The tree is initialized with $2n$ tasks: n regular tasks and n delayed tasks. The leaves are sorted by lct^0 for the regular tasks and lct^1 for the delayed tasks. For an arbitrary leaf v, the energy and envelopes are defined as

$$\mathrm{e}^k_{v^0} = \begin{cases} \mathrm{c}_v\, p^0_v & \text{if } k \geq 0 \\ 0 & \text{if } v^0 \notin \Theta \end{cases}$$

$$\mathrm{e}^k_{v^1} = \begin{cases} 0 & \text{if } k = 0 \\ \mathrm{c}_v\, d_v & \text{if } k > 0 \\ 0 & \text{if } v^1 \notin \Theta \end{cases}$$

$$\mathrm{Env}'^k_{v^0} = \begin{cases} \mathcal{C}\,\mathrm{lct}^0_v - \mathrm{e}^k_{v^0} & \text{if } k \geq 0 \\ \infty & \text{if } v^0 \notin \Theta \end{cases}$$

$$\mathrm{Env}'^k_{v^1} = \begin{cases} \infty & \text{if } k = 0 \\ \mathcal{C}\,\mathrm{lct}^1_v - \mathrm{c}_v\, d_v & \text{if } k > 0 \\ \infty & \text{if } v^1 \notin \Theta \end{cases}$$

$$\mathrm{e}^{\Lambda k}_{v^0} = \begin{cases} \mathrm{c}_v\, p^0_v & \text{if } (k \geq 0) \wedge (v^0 \in \Lambda^0) \\ -\infty & \text{if } v^0 \notin \Lambda^0 \end{cases}$$

$$\mathrm{e}^{\Lambda k}_{v^1} = \begin{cases} 0 & \text{if } (k = 0) \wedge (v^1 \in \Lambda^1) \\ (p^0_v + d_v)\,\mathrm{c}_v & \text{if } (k > 0) \wedge (v^1 \in \Lambda^1) \\ -\infty & \text{if } v^1 \notin \Lambda^1 \end{cases}$$

$$\mathrm{Env}'^{\Lambda k}_{v^0} = \begin{cases} \mathcal{C}\,\mathrm{lct}^0_v - \mathrm{c}_v\, p^0_v & \text{if } (k \geq 0) \wedge (v^0 \in \Lambda^0) \\ \infty & \text{if } v^0 \notin \Lambda^0 \end{cases}$$

$$\mathrm{Env}'^{\Lambda k}_{v^1} = \begin{cases} \infty & \text{if } (k=0) \wedge (v^1 \in \Lambda^1) \\ \mathcal{C}\,\mathrm{lct}^1_v - (p^1_v + d_v)\,\mathrm{c}_v & \text{if } (k>0) \wedge (v^1 \in \Lambda^1) \\ \infty & \text{if } v^1 \notin \Lambda^1 \end{cases}$$

Similar to the $(\Theta - \Lambda)^r_L-$tree, the superscript k stands for an upper bound on the number of delayed tasks in the leaf v. In order to make the computations of the $\Lambda-$energy and $\Lambda-$latest energy envelope symmetrical over all nodes of the tree, we suppose that $0 \leq k \leq r$ and since $\mathrm{e}^{\Lambda k}_v$ and $\mathrm{Env}'^{\Lambda k}_v$ are the $\Lambda-$energy and $\Lambda-$latest energy envelope when at most k tasks are delayed, for $k \geq 2$ in the leaves of the $(\Theta - \Lambda)^r_L-$tree we necessarily have $\mathrm{e}^{\Lambda k}_v = \mathrm{e}^{\Lambda 1}_v$ and $\mathrm{Env}'^{\Lambda k}_v = \mathrm{Env}'^{\Lambda 1}_v$.

Scheduling a regular or delayed task in $\Lambda^0 \cup \Lambda^1$ is equivalent to adding the task i^0 to Λ^0 or the task i^1 to Λ^1 and updating the node corresponding to the task according to the formulae above. Unscheduling a regular or delayed task from $\Lambda^0 \cup \Lambda^1$ is equivalent to removing the task i^0 from Λ^0 or the task i^1 from Λ^1 and updating the node corresponding to the task according to the formulae above. Unscheduling a task i from Θ removes both i^0 and i^1.

Let w be an internal node of the tree. Scott [**?**] proves that

$$\mathrm{Env}'^0_w = \min(\mathrm{Env}^0_{\mathrm{right}(w)} - \mathrm{e}^0_{\mathrm{left}(w)}, \mathrm{Env}'^0_{\mathrm{left}(w)}) \tag{51}$$

$\mathrm{Env}'^{\Lambda k}_w$ is the minimum $\Lambda-$latest energy envelope of the tasks in Θ whose leaves are descendant of w and to which one task from $\Lambda^0 \cup \Lambda^1$ is added. The task from $\Lambda^0 \cup \Lambda^1$ is also a descendant of w. With a reasoning similar to (28) and (36), the values of Env'^k_w and $\mathrm{Env}'^{\Lambda k}_w$ for at most k delayed tasks, $0 \leq k \leq r$, are recursively computed as

$$\mathrm{Env}'^k_w = \min_{0 \leq j \leq k} \{\mathrm{Env}'^j_{\mathrm{right}(w)} - \mathrm{e}^{k-j}_{\mathrm{left}(w)}\} \cup \{\mathrm{Env}'^k_{\mathrm{left}(w)}, \mathrm{Env}'^{(k-1)}_w\} \tag{52}$$

$$\mathrm{Env}'^{\Lambda k}_w = \min_{0 \leq j \leq k} \{\mathrm{Env}'^{\Lambda j}_{\mathrm{right}(w)} - \mathrm{e}^{k-j}_{\mathrm{left}(w)}, \mathrm{Env}'^j_{\mathrm{right}(w)} - \mathrm{e}^{\Lambda(k-j)}_{\mathrm{left}(w)}\} \cup \{\mathrm{Env}'^{\Lambda k}_{\mathrm{left}(w)}, \mathrm{Env}'^{\Lambda(k-1)}_w\} \tag{53}$$

In the case that the number of delayed tasks is greater than the number of available nodes in the subtree of the right side, it is sufficient to retrieve $\mathrm{Env}'^{\Lambda(k-1)}_w$ in (53), i.e. the lambda energy envelope when at most $k-1$ tasks are delayed.

Finally, the function $\mathrm{Env}'^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1)$ for $\Theta^0 = \{i^0 \mid i \in \Theta\}$ and $\Theta^1 = \{i^1 \mid i \in \Theta\}$ can be computed by computing the value $\mathrm{Env}'^{\Lambda k}_w$ at the root node of $(\Theta - \Lambda)^r_U-$tree.

4.2.5 Robust Edge-Finding algorithm for filtering the latest completion times

Analogous to the case for filtering the earliest starting times, the implementation of Edge-Finding for filtering the latest completion times proceeds in two phases.

Algorithm 6: DetectionPhaseOfEdge-FindingForUpperBounds($\mathcal{I}$)

```
1  for $i \in \{1, ..., n\}$ do
2      prec[$i$,0] $\leftarrow \infty$;
3      prec[$i$,1] $\leftarrow \infty$;
4  $\Theta \leftarrow \mathcal{I}$;
5  $\Lambda^0 \leftarrow \emptyset$;
6  $\Lambda^1 \leftarrow \emptyset$;
7  for $j \in \mathcal{I}$ in non-decreasing order of $\mathrm{est}_j$ do
8      while $\mathrm{Env}'^{\Lambda r}_{\mathrm{root}}(\Theta, \Lambda) < \mathcal{C} \cdot \mathrm{est}_j$ do
9          $i^b \leftarrow$ The task in $\Lambda^0 \cup \Lambda^1$ that minimizes $\mathrm{Env}'^{\Lambda r}(\Theta, \Lambda)$ ;
10         prec[$i, b$] $\leftarrow \mathrm{est}_j$;
11         $\Lambda^b \leftarrow \Lambda^b \setminus \{i^b\}$;
12     $\Theta \leftarrow \Theta \setminus \{j\}$;
13     $\Lambda^0 \leftarrow \Lambda^0 \cup \{j^0\}$;
14     $\Lambda^1 \leftarrow \Lambda^1 \cup \{j^1\}$;
```

Detection phase

Algorithm 6 adapts the detection phase of the Edge-Finding for the latest completion times. The algorithm starts with a full $(\Theta - \Lambda)^r_U-$tree, in which all the regular as well as the delayed tasks are scheduled in Θ and $\Lambda = \Lambda^0 \cup \Lambda^1$ is empty. That is,

$$\Theta = \mathcal{I}^0 \cup \mathcal{I}^1, \Lambda^0 = \Lambda^1 = \emptyset$$

The algorithm iterates over the set of all earliest starting times in non-decreasing order. If the filtering of earliest starting times and latest completion times of tasks are both implemented respectively, it is not necessary to test the Overload Checking in algorithm 6. For every $j \in \mathcal{I}$ in non-decreasing order

$$\mathrm{Env}'^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1) < \mathcal{C} \cdot \mathrm{est}_j \tag{54}$$

is checked which captures the precedence. Thanks to the structure of the $(\Theta - \Lambda)^r_U-$tree, $\mathrm{Env}'^{\Lambda r}(\Theta^0, \Theta^1, \Lambda^0, \Lambda^1)$ is retrieved from the root of $(\Theta - \Lambda)^r_U-$tree for each $j \in \mathcal{I}$. Line 9 retrieves the task subject to a precedence. It can be implemented in $O(\log(n))$ time by traversing down the tree. The algorithm proceeds by traversing down the $(\Theta - \Lambda)^r_U-$tree from the root. At each inner node w in such a traversal, the algorithm determines which one of the cases in (53) satisfy for $\mathrm{Env}'^{\Lambda k}_w$. So long as this task is taken from an $\mathrm{Env}'^{\Lambda k}_f$, where f is a child of w, the algorithm continues down. As soon as the task is taken from an $\mathrm{e}^{\Lambda k}_f$, the algorithm switches to check the cases of (35). Similarly, the responsible task subject to a precedence is only one task $i^b \in \Lambda^b$ that is located on a leaf of the $(\Theta - \Lambda)^r_U-$tree and causes (50) to be minimized, hence making a precedence. We encode the precedence that i creates in a two dimensional matrix by 0 and 1 columns, named prec, which indicates whether the task is regular or delayed (line 10). For $b \in \{0, 1\}$, $prec[i, b] = \mathrm{est}_j$ for some $j \in \mathcal{I}$ means that the subsets of tasks from $\Theta^0 \cup \Theta^1$ with at most $r - b$ delayed tasks which start no earlier than j are preceded by i^b and i^b cause (50) to be minimized. Once the precedence is recorded at line 10, the algorithm unschedules i^b from Λ^b. After the execution of the

b / Task	0	1
A	6	6
B	15	15
C	∞	∞
D	22	22

Table 5 The *prec* matrix which is obtained after the execution of the algorithm 6 on the instance of example 3.

loop, the tasks corresponding to j is unscheduled from Θ and rather scheduled in Λ^0 and Λ^1.

Figure 4 depicts a trace of the algorithm for the example 3 in few steps. Figure 4a illustrates the initialization step of the algorithm, where the $(\Theta - \Lambda)^r_U-$tree is full. That is, all the regular as well as delayed tasks are scheduled and no task is a candidate yet to create a precedence.

In the first iteration the task A is processed and it gets unscheduled. Thus, the two leaves of the $(\Theta - \Lambda)^r_U-$tree corresponding to A^0 and A^1 are modified. The updated status of the node corresponding to A is depicted in figure 4b. In the second iteration, B is processed. At this point, $30 < \mathcal{C} \cdot \mathrm{est}_B$, which causes the loop dedicated to detect an existing precedence to execute. Since A^1 is detected as the responsible task, it gets unscheduled. The loop executes again and A^0 is detected as responsible task. Therefore, it gets unscheduled, as illustrated in the figure 4c. In the next execution, the loop gets pasted and now B which corresponds to the time point t is unscheduled. Figure 4d represents this step. Table 5 indicates the prec matrix as the result of the algorithm 6.

Adjustment phase

For $j \in \mathcal{I}$, we define the *right cut* of j with respect to its est_j to be

$$\mathrm{Rcut}(j) = \{l \in \mathcal{I} : \mathrm{est}_j \leq \mathrm{est}_l\}$$

i.e. $\mathrm{Rcut}(j)$ includes all of the tasks which start no earlier than j.

The adjustment of the latest completion times is done by iterating over the detected precedences which are recorded in the prec matrix. Let i^b be the task for which a precedence was detected with (47) or (48) and $\mathrm{prec}[i, b] = \mathrm{est}_j$ for some $j \in \mathcal{I}$. The following formula adjusts the latest completion time of i.

$$\mathrm{lct}^0_i \leftarrow \min(\mathrm{lct}^0_i, \min_{\substack{\Omega^0 : \mathcal{I}(\Omega^0) \subseteq \mathrm{Rcut}(j) \setminus \{i\} \\ \Omega^1 : \mathcal{I}(\Omega^1) \subseteq \mathrm{Rcut}(j) \setminus \{i\} \\ \emptyset \neq \Omega^0 \cup \Omega^1 \\ |\Omega^1| \leq r - b \\ \mathrm{rest}(\Omega^0, \Omega^1, \mathrm{c_i}) > 0}} \{\max(\mathrm{lct}^0_{\mathcal{I}(\Omega^0)}, \mathrm{lct}^1_{\mathcal{I}(\Omega^1)}) - \left\lceil \frac{\mathrm{rest}(\Omega^0, \Omega^1, \mathrm{c}_i)}{\mathrm{c}_i} \right\rceil \}) \tag{55}$$

Note that in (55), Ω^0 includes the regular tasks and Ω^1 includes the delayed tasks as described in section 4.2.4.

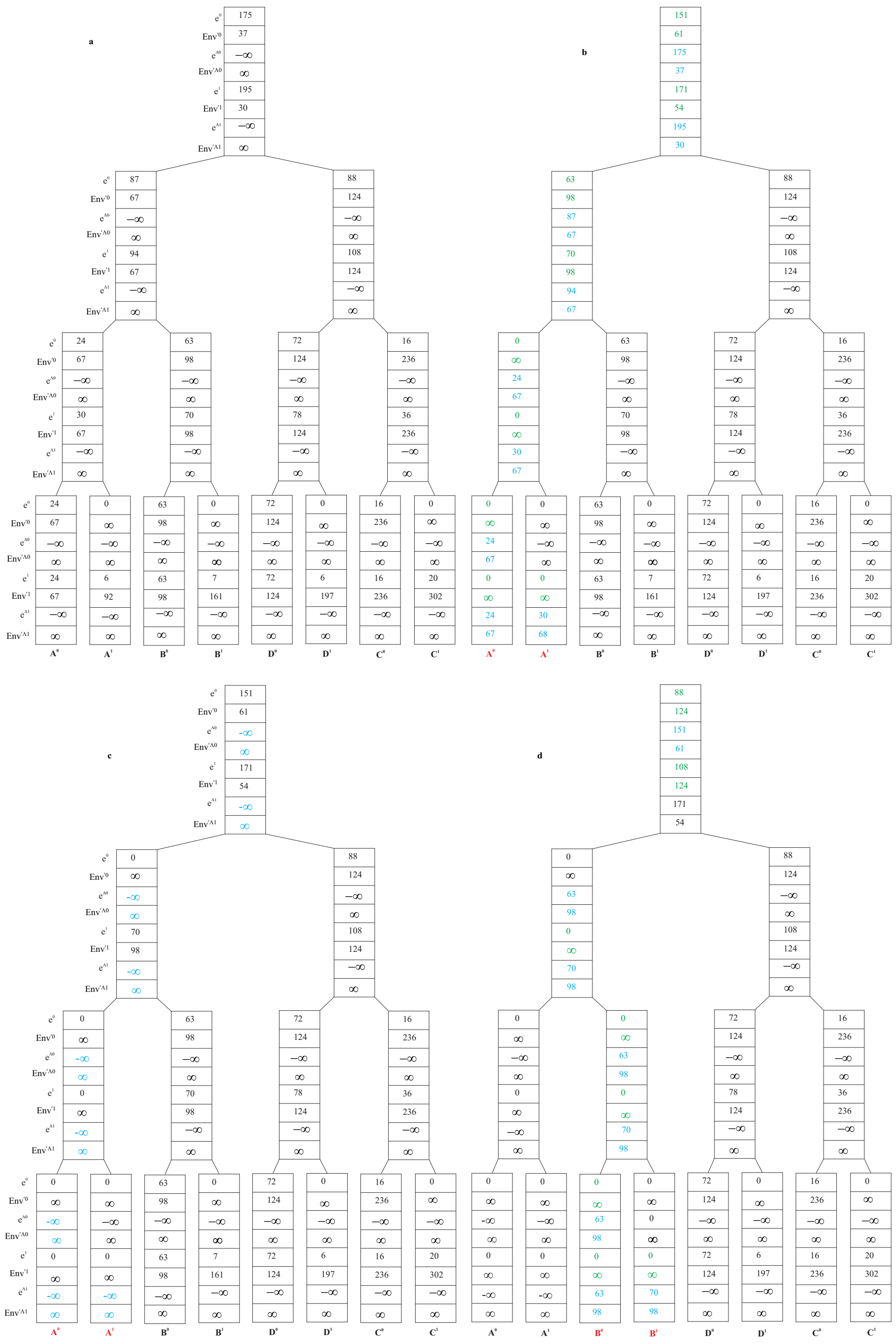


Fig. 4 In the figure 4a, the algorithm starts with a full $(\Theta-\Lambda)^r_U-$tree. In the figure 4b, A gets unscheduled from Θ and scheduled in Λ. In the figure 4c, A^1 and then A^0 are detected as responsible tasks and get unscheduled. In the figure 4d, B is unscheduled.

Algorithm 7: AdjustmentOfUpperBounds(prec,r,$\mathcal{C}$)

```
$F \leftarrow$ The capacities of the tasks for which a precedence was detected;
for c $\in F$ do
    $\Theta \leftarrow \emptyset$;
    upd $\leftarrow [\infty, \infty]$;
    $t' \leftarrow \infty$;
    for $i^b \in \{j^b \mid prec[j,b] < \infty \wedge \mathrm{c}_i = \mathrm{c}\}$ in non-increasing order of prec[i, b] do
        $t \leftarrow \mathrm{prec}[i,b]$ for some $j \in \mathcal{I}$;
        $\Theta \leftarrow \Theta \cup (\mathrm{Rcut}(j) \setminus \mathrm{Rcut}(t')) \setminus \{i\}$ ;
        $v \leftarrow \mathrm{MinIct}(\Theta, t, c, \mathcal{C}, r, b)$;
        $\mathrm{Env}' \leftarrow \mathrm{EnvelopeForUpperBound}(v, \Theta, r - b)$;
        diff $\leftarrow \lfloor (\mathrm{Env}' - (\mathcal{C} - \mathrm{c})t / \mathrm{c} \rfloor$;
        upd[b] $\leftarrow$ min(upd[b],diff);
        $\mathrm{lct}_i^0 \leftarrow \min(\mathrm{lct}_i^0$,upd[$b$]);
        if $\mathrm{est}_i \geq t$ then
            $\Theta \leftarrow \Theta \cup \{i\}$
        $t' \leftarrow t$;
```

From the condition $\mathrm{rest}(\Omega^0, \Omega^1, \mathrm{c}_i) > 0$ we obtain

$$(\mathcal{C} - \mathrm{c}_i)(\mathrm{est}_{\Omega^0 \cup \Omega^1}) > (\mathcal{C} - \mathrm{c}_i)(\max(\mathrm{lct}^0_{\mathcal{I}(\Omega^0)}, \mathrm{lct}^1_{\mathcal{I}(\Omega^1)})) - \mathrm{e}_{\Omega^0 \cup \Omega^1} \tag{56}$$

The right side of (56) describes an equivalent of $\mathrm{Env}'^r(\Theta^0, \Theta^1)$, defined by (49), on a resource with capacity $\mathcal{C} - \mathrm{c}_i$. This motivates the idea of defining a variant of the latest energy envelope, defined as (5), with respect to each $c \in F$ as

$$\mathrm{Env}'^{\mathrm{c}}_{\Theta} = \min_{\Omega \subseteq \Theta} ((\mathcal{C} - \mathrm{c})\,\mathrm{lct}^0_{\Omega} - \mathrm{e}_{\Omega}) \tag{57}$$

and a variant of $\mathrm{Env}'^r(\Theta^0, \Theta^1)$, defined in (49), as

$$\mathrm{Env}'^{\mathrm{c}\,r}(\Theta^0, \Theta^1) = \min_{\substack{\Omega^0 \subseteq \Theta^0 \\ \Omega^1 \subseteq \Theta^1 \\ |\Omega^1| \leq r \\ \mathcal{I}(\Omega^0) \cap \mathcal{I}(\Omega^1) = \emptyset}} ((\mathcal{C} - \mathrm{c}) \max(\mathrm{lct}^0_{\mathcal{I}(\Omega^0)}, \mathrm{lct}^1_{\mathcal{I}(\Omega^1)}) - \mathrm{e}_{\Omega^0} - \mathrm{e}_{\Omega^1}) \tag{58}$$

The algorithm 7 is our adaption of the adjustment phase of the Edge-Finding for this purpose. For each $\mathrm{c} \in F$, we initialize an empty $\Theta^{\mathrm{c}\,r}_U-$tree. This tree is different from $\Theta^{\mathrm{c}\,r}_L-$tree and the structure of its leaves is similar to $(\Theta - \Lambda)^r_U-$tree. That is, there are $2n$ leaves in the tree, associated to the regular and delayed tasks. Each leaf of the $\Theta^{\mathrm{c}\,r}_U-$tree holds the parameters for the energy and latest energy envelope, respectively defined by (3) and (5), as well as (57) which is computed for $\Theta = \{i\}$. For an inner node w of the $\Theta^{\mathrm{c}}-$tree, (57) is computed for the set of all tasks which are included in the subtree rooted at w. This computation is done recursively as below.

$$\mathrm{Env}'^{\mathrm{c}\,k}_w = \min_{0 \leq j \leq k}(\mathrm{Env}'^{\mathrm{c}\,j}_{\mathrm{right}(w)} - \mathrm{e}^{k-j}_{\mathrm{left}(w)}, \mathrm{Env}'^{\mathrm{c}\,k}_{\mathrm{left}(w)}, \mathrm{Env}'^{\mathrm{c}(k-1)}_w) \tag{59}$$

$\Theta^{\mathrm{c}\,r}_U-$tree develops by processing all $\mathrm{prec}[i,b]$ in non-increasing order. The adjustment should be done only if $\mathrm{c}_i = \mathrm{c}$. If so, at iterating each $\mathrm{prec}[i,b] = \mathrm{est}_j$ for

Algorithm 8: Minlct(tree,bound,c,$\mathcal{C}$,k,b)

```
$v \leftarrow$ root;
e[0...$k$] $\leftarrow$ **0** ;
minEnv$^c \leftarrow (\mathcal{C} - \mathrm{c})$.bound;
$k \leftarrow k - b$;
**while** $v$ is not a leaf **do**
    branchLeft $\leftarrow$ false;
    **for** $j = 0, ..., k$ **do**
        **if** $\mathrm{Env}'^{c,j}_{\mathrm{left(v)}} - \mathrm{e}[k-j] <$minEnv$^c$ **then**
            $v \leftarrow \mathrm{left(v)}$;
            branchLeft $\leftarrow$ true;
            **break**;
    **if** not branchLeft **then**
        $v \leftarrow \mathrm{right(v)}$;
        **for** $j = 0, ..., k$ **do**
            $\mathrm{e}'[j] \leftarrow \max_{0 \le i \le j}(\mathrm{e}[i] + \mathrm{e}^{j-i}_{\mathrm{left(v)}}, \mathrm{e}'[j-1])$
        $\mathrm{e} \leftarrow \mathrm{e}'$;
**return** $v$;
```

some $j \in \mathcal{I}$, all the tasks which cannot start before j are scheduled in the $\Theta^{\mathrm{c}\,r}_U-$tree by adding their regular version to Θ^0 and their delayed version to Θ^1.

Let $\Omega^0 \cup \Omega^1$ be the candidate subset which can give the strongest adjustment in (55). Once all the tasks which are eligible to be in $\Omega^0 \cup \Omega^1$ are scheduled in the tree, it is time to identify Ω^0 and Ω^1 and compute the value of the second component of the right side of (55). We proceed to compute Ω^0 and Ω^1 in two steps.

The first step is to compute the minimum of $\max(\mathrm{lct}^0_{\mathcal{I}(\Omega^0)}, \mathrm{lct}^1_{\mathcal{I}(\Omega^1)})$ (or *minlct*) for all Ω^0 and Ω^1 that satisfy (55). In the algorithm 8, we consider this idea by taking advantage of the new variant of the latest energy envelope to locate such a node.

The second step is to compute the envelope of the tasks which is done in the algorithm 9 . This is similarly done by dividing the tasks in two groups $\alpha(j, \mathrm{c})$ and $\beta(j, c)$, for the tasks that finish at or after minlct and the tasks which finish before minlct. From the sets $\alpha(j, \mathrm{c})$ and $\beta(j, c)$, the latest energy envelop in the $\Theta^{\mathrm{c}\,r}_U-$tree is computed with

$$\mathrm{Env}(j, c) = \mathrm{Env}_{\alpha(j,\mathrm{c})} - \mathrm{e}_\beta \tag{60}$$

where e_β and Env_α are computed in a bottom-up manner, starting from the located task responsible for minlct, and by taking into account that at most r tasks are delayed. Once the envelope is computed, the value which makes the strongest update is adjusted by taking the minimum between the envelope computed in the current iteration and the preceding iterations (line 12 of the algorithm 7). Ultimately, if the task can not start before est_j, it gets scheduled itself.

Finally, it must be mentioned that the same complexity for filtering latest completion times is maintained and the argument is similar.

Algorithm 9: EnvelopeForUpperBound(v,tree,bound,c,k)

```
$e_\alpha \leftarrow [e_v^0, e_v^1, 0, ..., 0]$; // $e_\alpha$ includes $k$ - 1 entries 0
$Env'_\alpha \leftarrow [Env_v^0, Env_v^1, \infty, ..., \infty]$; // $Env_\alpha$ includes $k$ - 1 entries $\infty$
$e_\beta \leftarrow \mathbf{0}$ ;
while $v$ is not the root do
    if $v$ is a right child then
        $e'_\beta \leftarrow \mathbf{0}$ ;
        for $j = 0, ..., k$ do
            $e'_\beta[j] \leftarrow \max_{0 \le i \le j}(e_\beta[i] + e^{j-i}_{\mathrm{sibling}(v)}, e'_\beta[j-1])$
        $e_\beta \leftarrow e'_\beta$;
    else
        $Env''_\alpha \leftarrow \mathbf{0}$ ;
        $e'_\alpha \leftarrow \mathbf{0}$ ;
        for $j = 0, ..., k$ do
            $Env''_\alpha[j] \leftarrow \min(\min_{0 \le i \le j}(Env'^{i}_{\mathrm{sibling}(v)} - e_\alpha[j-i]), Env'_\alpha[j], Env''_\alpha[j-1])$;
            $e'_\alpha[j] \leftarrow \max(\max_{0 \le i \le j}(e^i_{\mathrm{sibling}(v)} + e_\alpha[j-i]), e'_\alpha[j-1])$;
    $Env'_\alpha \leftarrow Env''_\alpha$;
    $e_\alpha \leftarrow e'_\alpha$;
    $v \leftarrow \mathrm{parent}(v)$;
return $\min_{0 \le i \le k}(Env'_\alpha[k-i] - e_\beta[i])$;
```

5 Experiments

The experiments were carried out on a 2.0 GHz Intel Core i5, with Choco version 3.3.1. We tested our algorithms against the BL suite of the RCPSP instances [?]. This benchmark consists of 40 highly cumulative instances with either 20 or 25 tasks, subject to precedence constraints, to be executed on several resources. We minimize the makespan. For the delay attributes associated to every task i, we uniformly generated random numbers in $[0, 2 \cdot p_i^0]$. We used three different heuristics: Lexicographic, DomOverWDeg [?], and Impact Based Search [?]. For these three heuristics, the figures 5, 6 and 7 respectively illustrate a logarithmic scale representation of the number of backtracks and the elapsed time measurements when a combination of time-tabling and overload checking or time-tabling and Edge-Finding are implemented. For the time-tabling algorithm, we used the same implementation as in Derrien et al. [?] that we obtained from the authors.

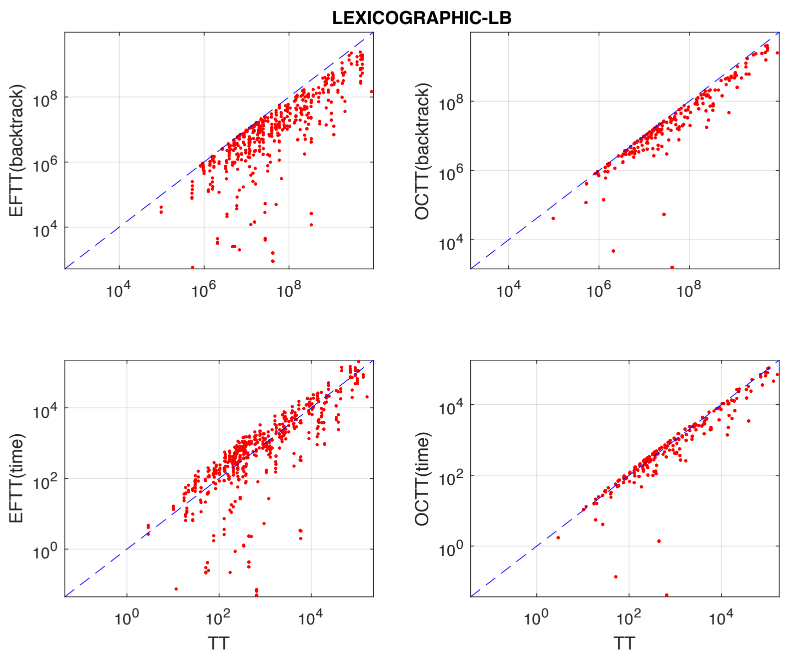


Fig. 5 The logarithmic scale graphs as the results of running time-tabling and overload checking (denoted OC-TT) or time-tabling and Edge-Finding(denoted EF-TT) with lexicographic heuristics in terms of backtrack numbers as well as elapsed times. The horizontal axis corresponds to the time-tabling and the vertical axis corresponds to overload checking or Edge-Finding.

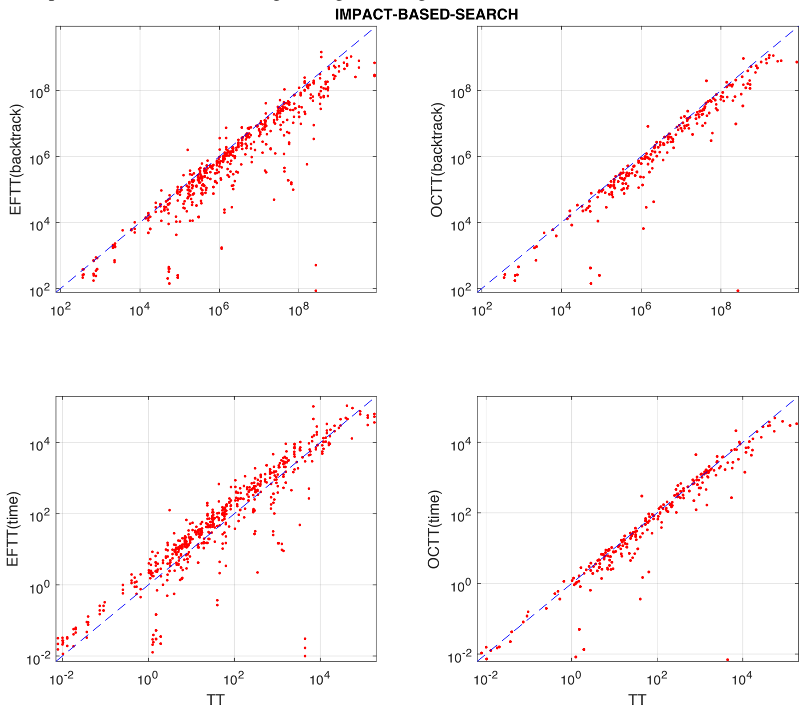


Fig. 6 The logarithmic scale graphs as the results of running time-tabling and overload checking (denoted OC-TT) or time-tabling and Edge-Finding(denoted EF-TT) with impact based search heuristics in terms of backtrack numbers as well as elapsed times. The horizontal axis corresponds to the time-tabling and the vertical axis corresponds to overload checking or Edge-Finding.

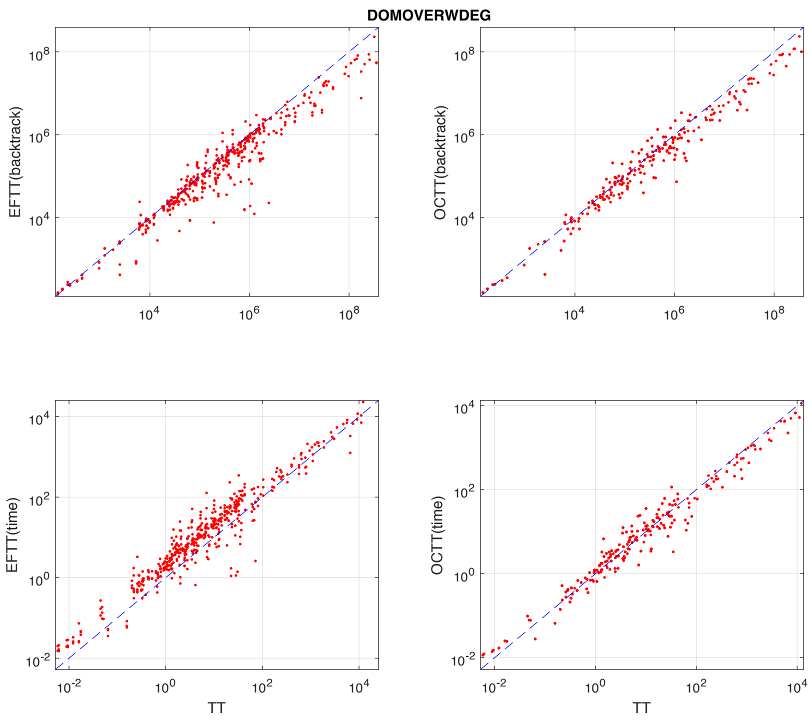


Fig. 7 The logarithmic scale graphs as the results of running time-tabling and overload checking (denoted OC-TT) or time-tabling and Edge-Finding(denoted EF-TT) with domoverwdeg heuristics in terms of backtrack numbers as well as elapsed times. The horizontal axis corresponds to the time-tabling and the vertical axis corresponds to overload checking or Edge-Finding.

As the graphs verify, our algorithms lead to fewer backtracks. The results are much more significant for the Edge-Finding. This is due to the fact that the Edge-Finding filters the domains while the overload checking only triggers backtracks. It appears that the heuristic chosen for solving the problem also affects the results. Thereby, we selected the lexicographic heuristic as one of our heuristics so that the heuristic chosen does not impinge our objective, which is proving that the combination of our algorithms with the time tabling provides a stronger filtering. The lexicographic heuristic is certainly not the best heuristic to solve scheduling problems, but it allows seeing how much pruning a new filtering algorithm achieves. The combination of our algorithms with time tabling improves the resolution times for many instances. This fact can differ from one heuristic to another. Overall, among the state of the art heuristics, IMPACT-BASED-SEARCH performs more efficiently, as it leads to fewer backtracks and faster computation times for many instances.

We also compared the implementation of our Edge-Finding algorithm with a conjunction of n CUMULATIVE constraints, as described in section 2.2. The performances for the conjunction of n CUMULATIVE constraints are so much worse in terms of time that we omit to report them.

6 Conclusion

We adapted the state of the art algorithms for overload checking and Edge-Finding in robust cumulative scheduling problems. The experimental results demonstrate a stronger filtering when our algorithms are combined with time-tabling.

Acknowledgments

The authors would like to thank Alban Derrien for providing his implementation of the time tabling algorithm [**?**].

The SVJOUR3 document class users guide Version 3.2 – for Springer journals

8 May 2007

Contents

1 Introduction

This documentation describes the SVJOUR3 LaTeX 2_ε document class. It is not intended to be a general introduction to TeX or LaTeX. For this we refer to [2] and [3].

SVJour3 was derived from the LaTeX 2_ε `article.cls`, based on TeX version 3.141 and LaTeX 2_ε. Hence text, formulas, figures and tables are typed using the standard LaTeX 2_ε commands. The standard sectioning commands are also used.

The main differences to the standard article class are the presence of additional high-level structuring commands for the article header, new environments for theorem-like structures, and some other useful commands.

Please always give a `\label` where possible and use `\ref` for cross-referencing. Such cross-references will be converted to hyper-links in the electronic version. The `\cite` and `\bibitem` mechanism for bibliographic references is also obligatory.

1.1 Overview

The documentation consists of this document—which describes the whole class (i.e. the differences to the `article.cls`)—and a ready-to-use template to allow you to start writing immediately.

1.2 Using PostScript fonts

Springer journals produced in TeX are typeset using the PostScript[1] Times fonts for the main text. As the use of PostScript fonts results in different line and page breaks than when using Computer Modern (CM) fonts, we encourage you to use our document class together with the `psnfss` package `mathptmx`. This package makes all the necessary font replacements to show you the page make-up nearly as it will be printed. Ask your local TeXpert for details. PostScript previewing is possible on most systems. On some installations, however, on-screen previewing may be possible only with CM fonts.

If, for technical reasons, you are not able to use the PS fonts, it is also possible to use our document class together with the ordinary Computer Modern fonts. Note, however, that in this case line and page breaks will change when we reTeX your file with PS fonts, making it necessary for you to check them again carefully once you receive the proofs from the printer.

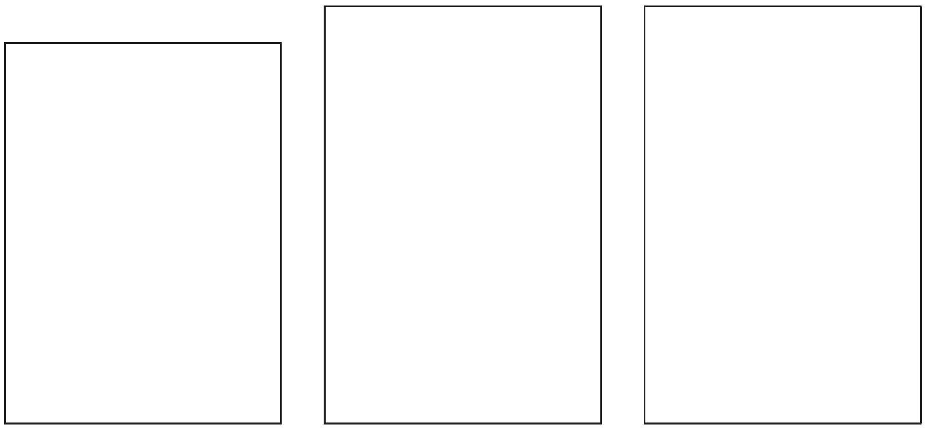
2 Initializing the class

To use the document class, enter

[1]PostScript is a trademark of Adobe.

```
\documentclass [⟨format,other options⟩] {svjour3} [⟨release-date⟩]
```

at the beginning of your article. The first option `[`⟨*format*⟩`]` is required and should be set according to the journal for which you are planning to submit a contribution. Three formats are available. The format is pre-set in the template, but choose the one that suits your specific journal if there is no journal-specific template available for your journal.



`[twocolumn]` `[smallcondensed]` (default) `[smallextended]`

available `format` options (samples not in full-scale)

There is one general option `[`⟨*glov3*⟩`]` that is auto-activated if no special option for the particular journal exists or is given. This option causes LaTeX to read in the class option file `svglov3.clo` (part of the package). Do not try to use those options of the old SVJour classes version 1 and 2 as these are not suitable for SVJour3—you will get a class error, tops.

Other options, valid for every journal, are

draft	to make overfull boxes visible,
final	the opposite, and
referee	required to produce a hardcopy for the referee with a special layout (bigger interline spacing).

The next four additional options control the automatic numbering of figures, tables, equations, and theorem-like environments. The fifth option described below disables the "Springer" theorems (see also Sect. 5). The last option describes the natbib package.

numbook	"numbering like the standard book class"—prefixes all the numbers mentioned above with the section number,
envcountsect	the same for theorem like environments only,
envcountsame	uses one counter for all theorem-like environments,
envcountreset	resets the theorem counter(s) every new section,

nospthms	use it *only* if you want to suppress all Springer theorem-like environments (see Sect. 5) and use the theorem environments of original LaTeX package or other theorem packages instead. (Please check this with your editor.)
natbib	handles reference entries in the author-year system (with or without BibTeX) by using the natbib package of Patrick W. Daly. It can be found at the *Comprehensive TeX Archive Network* (CTAN...tex-archive/macros/latex/contrib/supported/natbib/), see [4, 5, 6].

If a journal contains articles in languages other than English the class provides two options "`[deutsch]`" and "`[francais]`" that automatically translate supplied texts or phrases given from LaTeX.

There may be additional options for a specific journal—please refer to the extra documentation or to the template file.

As an example, we show how to begin a document for a two-column journal produced in draft mode:

```
\documentclass[twocolumn,draft]{svjour3}
```

3 The article header

In this section we describe the usage of the high-level structuring commands for the article header. Header in this context means everything that comes before the abstract.

3.1 The title

The commands for the title and subtitle of your article are

```
\title {⟨your title⟩}
\subtitle {⟨your subtitle⟩}
```

You can also dedicate your article to somebody by specifying

```
\dedication{⟨dedication⟩}
```

3.2 Authors

Information about the authors is provided with

`\author {⟨author name [\and author name] ⟩}`

If there is more than one author, the names should be separated by `\and`. To make this clear, we provide an example:

```
\author{John B. Doe \and Sally Q. Public \and Joe A. Smith}
```

3.3 Addresses

Address information is marked with

`\institute {⟨address information [\and address information]⟩}`

If there is more than one address, the entries are separated by `\and`.

As the address of the author appears as a footnote on the first page of your article, the author name is to be repeated in the address information with an `\at` depicting the affiliation. Addresses should be contained in one line, using commas to separate the parts of the address. In addition, you can use

`\email {⟨email address⟩}`

to provide an email address within `\institute`.

If there are authors appearing with different addresses the affiliations can be indicated with the same author listed "`\at`" (i.e. before) each particular address in the `\institute{...}` field—authors in such lists (read: at the same address) should again be separated by an `\and`.

To continue the example above, we could say

```
\institute{J.B. Doe
           \at Doe Institute, 281 Prime Street, Daisy Town, NA 02467,
               USA\\Tel.: +127-47-678901, Fax: +127-47-678907
        \and
        J.B. Doe \and S.Q. Public
           \at Public-Enterprises
        \and
        J.A. Smith
           \at Smith University,\\\email{smith@smith.edu}
        }
```

3.4 Footnotes to the title block

If footnotes to the title, subtitle, author's names or institute addresses are needed, please code them with

```
\thanks {⟨text of footnote⟩}
```

immediately after the word in the corresponding field. Please note that these footnotes are not marked—they will appear above the address information at the bottom of the first page, enclosed in rules.

3.5 Changing the running heads

Normally the running heads—if present in the specific journal—are produced automatically by the `\maketitle` command using the contents of `\title` and `\author`. If the result is too long for the page header (running head) the class will produce an error message and you will be asked to supply a shorter version. This is done using the syntax

```
\titlerunning{⟨text⟩}
\authorrunning{⟨first author et al.⟩}
```

These commands must be entered before `\maketitle`.

3.6 Typesetting the header

Having entered the commands described in this section, please format the heading with the standard `\maketitle` command. If you leave it out, the work done so far will produce *no* text.

4 Abstract, keywords, MSC, PACS, and CR codes

The environment for the abstract is the same as in the standard article class. To insert keywords, a “Mathematics Subject Classification” (MSC), “Physics and Astronomy Classification Scheme” (PACS), or “ACM Computing Classification” (CR) codes you should use

```
\keywords{⟨keywords⟩}
\subclass{⟨MSC codes⟩}
\PACS{⟨PACS codes⟩}
\CRclass{⟨CR codes⟩}
```

at the end—but still inside—of the abstract environment; the individual words or codes should be separated by `\and`.

Some journals published in other languages than English reapeat those elements in translation at the end of the header material before the actual article starts.

Please use the following environment for that and give the relevant codes (MSC, PACS, CR) only in the translated abstract (see also the particular template file)

```
\begin{translation}{english}
  \begin{abstract}
    ...
  \end{abstract}
\end{translation}
```

5 Theorem-like structures

5.1 Predefined environments

In the SVJour3 document class the functions of the standard `\newtheorem` command have been enhanced to allow a more flexible font selection. All standard functions though remain intact (e.g. adding an optional argument specifying additional text after the environment counter). To typeset environments such as definitions, theorems, lemmas or examples, we have predefined the environments in the list below. Note that the font selection of environment heading vs. its body font is depicted in this list with

environment name	=	**bold heading**	*italic text body*
environment name	=	**bold heading**	normal text body
environment name	=	*italic heading*	normal text body

Unnumbered environments will be produced by:
claim and *proof*.

Numbered environments will be produced by:
theorem, ***proposition***, ***lemma***, ***corollary***,
definition, **exercise**, **problem**, **solution**,
remark, *note*, *case*, *conjecture*, *example*, *property*, and *question*.

The syntax is exactly the same as described in [3, Sect. 3.4.3]:

```
\begin{⟨environment⟩}[⟨name⟩]
...
\end{⟨environment⟩}
```

where the optional *name* is often used for the common name of the theorem:

```
\begin{theorem}[Church, Rosser]
...
\end{theorem}
```

Sometimes the automatic braces around the optional argument are unwanted (e.g. when it consists only of a reference made with `\cite`). Then you can wrap the whole theorem-like structure in a `theopargself` environment. It suppresses the braces and gives you complete control over the optional argument, e.g.:

```
\begin{theopargself}
   \begin{theorem}[\cite{Church,Rosser}]
   ...
   \end{theorem}
\end{theopargself}
```

5.2 Defining new structures

For cases where you do not find an appropriate predefined theorem-like structure above, we provide two mechanisms to define your own environment. Use

```
\spdefaulttheorem{⟨env name⟩}{⟨caption⟩}{⟨cap font⟩}{⟨body font⟩}
```

to define an environment compliant with the selected class options (see Sect. 2) and designed as the predefined Springer theorem-like environments.

Continuative commands you can use here are

```
\spnewtheorem{⟨name⟩}{⟨label text⟩}[⟨numbered within⟩]{⟨label font⟩}{⟨body font⟩}
\spnewtheorem{⟨name⟩}[⟨numbered like⟩]{⟨label text⟩}{⟨label font⟩}{⟨body font⟩}
```

There is also a starred version, without optional arguments, which provides a theorem environment without numbers. Here *name* is the name of the environment, *label text* is the text to be typeset as heading, and the *label font* and *body font* are the font for the label text and the theorem body.

If you use the *numbered within* argument, the new structure will be numbered within the specified sectional unit—if you specify *numbered like*, it shares its numbering sequence with the referenced structure.

For instance, the predefined environments *theorem* and *proof* are defined as

```
\spnewtheorem{theorem}{Theorem}{\bf}{\it}
\spnewtheorem*{proof}{Theorem}{\it}{\rm}
```

whereas one could define a theorem-like structure *algorithm*, numbered within the current section as

```
\spnewtheorem{algorithm}{Algorithm}[section]{\bf}{\rm}
```

It is also possible to skip all theorem features of the SVJour3 document class (see Sect. 2) and/or to use the *theorem* package shipped with LaTeX 2_ε (see [1] for a complete description) or the *amsthm* package of AMSLaTeX to define new theorem environments. But note that once you use them you should not change the predefined structures.

6 Additional commands

We provide some additional useful commands which you can use in your manuscript. The first is the *acknowledgements* environment

```
\begin{acknowledgements}
...
\end{acknowledgements}
```

which is usually used as the last paragraph of the last section.

The next is an enhancement of the standard `\caption` command used inside of *figure* environments to produce the legend. The added command

```
\sidecaption
```

can be used to produce a figure legend beside the figure. To activate this feature you have to enter it as the very first command inside the *figure* environment

```
\begin{figure}\sidecaption
  \resizebox{0.3\hsize}{!}{\includegraphics*{figure.eps}}
  \caption{A figure}
\end{figure}
```

If there is not enough room for the legend the normal `\caption` command will be used. Also note that this works only for captions that come *after* the included images.

We also have enhanced the *description* environment by an optional parameter, which lets you specify the largest item label to appear within the list. The syntax now is

```
\begin{description}[⟨largelabel⟩]
...
\end{description}
```

The texts of all items are indented by the width of *largelabel* and the item labels are typeset flush left within this space. Note: The optional parameter will work only two levels deep.

The often missed command

```
\qed
```

yields the known □ symbol with appropriate spacing to close e.g. a proof, use the new declaration

```
\smartqed
```

to move the position of the predefined qed symbol to be flush right (in text mode). If you want to use this feature throughout your article the declaration must be set in the preamble; otherwise it should be used individually in the relevant environment, i.e. proof. □

The last two commands working as markup in

```
\vec{⟨symbol⟩}
\tens{⟨symbol⟩}
```

mark vectors (e.g. $\boldsymbol{S}$, or $\mathbf{S}$) and tensors (e.g. $\mathsf{\mathbf{S}}$) respectively.

References

1. Mittelbach F., Goossens, M.: The LaTeX Companion, 2nd edn. Addison-Wesley, Boston, Massachusetts (2004)
2. Knuth D.E.: The TeXbook (revised to cover TeX3). Addison-Wesley, Reading, Massachusetts (1991)
3. Lamport L.: LaTeX: A Document Preparation System, 2nd edn. Addison-Wesley, Reading, Massachusetts (1994)
4. TeX Users Group (TUG), http://www.tug.org
5. Deutschsprachige Anwendervereinigung TeX e.V. (DANTE), Heidelberg, Germany, http://www.dante.de
6. UK TeX Users' Group (UK-TuG), http://uk.tug.org