Strong Polynomiality of Resource Constraint Propagation

Luc Mercier
mercier@cs.brown.edu

Research Comp Report
November 2005

Advisor: Pascal Van Hentenryck
pvh@cs.brown.edu
Comitee: Eli Upfal, Meinolf Sellman
{eli, sello}@cs.brown.edu
Brown University

Abstract

Constraint-based schedulers have been widely successful to tackle complex, disjunctive and cumulative, scheduling applications by combining tree search and constraint propagation. The constraint-propagation step is a fixpoint algorithm that applies pruning operators to tighten the release and due dates of activities using precedence or resource constraints. A variety of pruning operators for resource constraints have been proposed: they are based on edge finding or energetic reasoning and handle a single resource.

Complexity results in this area are only available for a single application of these pruning operators, which is problematic for at least two reasons. On the one hand, the operators are not idempotent so a single application is rarely sufficient. On the other hand, the operators are not used in isolation but interact with each other. Existing results thus provide a very partial picture of the complexity of propagating resource constraints in constraint-based scheduling.

This paper aims at addressing these limitations. It studies, for the first time, the complexity of applying pruning operators for resource constraints to a fixpoint. In particular, it shows that (1) the fixpoint of the edge finder for both release and due dates can be reached in strongly polynomial time for disjunctive scheduling; (2) the fixpoint can be reached in strongly polynomial time for updating the release dates or the due dates but not both for the cumulative scheduling; (3) the fixpoint of "reasonable" energetic operators cannot be reached in strongly polynomial time, even for disjunctive scheduling and even when only the release dates or the due dates are considered.

1 Introduction

Constraint-based schedulers (e.g., [AB93, BLPN01, CL94, LP94, MS96]) are widely successful in tackling complex, disjunctive or cumulative, scheduling problems in manufacturing, transportation, supply-chain management, and the steel industry. These problems often consist of minimizing the completion time of a set of jobs, each job being a sequence of tasks linked by precedence constraints. Each task has a processing time and may require some units of one or more resources. Disjunctive resources have a capacity one and two tasks requiring the same disjunctive resource cannot overlap in time. Cumulative resources have a finite capacity C and the total demand for a cumulative resource at any time t cannot exceed C.

Constraint-based schedulers approach the solving of complex scheduling problems by iterating two main steps until a feasible or optimal solution is found:

1. a constraint propagation step that tightens the release and the due dates of each activity;

2. a (nondeterministic) branching step that adds new precedence constraints or assigns a starting date to some activity.

This paper focus on the constraint propagation step that tightens the release and the due dates of each activity. It features a propagation algorithm that applies pruning operators until a fixpoint is reached, i.e., until no further tightenings of the release and due dates is possible. The pruning operators typically focus on a single resource in isolation and are generally based on edge finding or energetic reasoning. Moreover, because even the one-resource problems are NP-complete in presence of release and due dates, these pruning operators apply tightening rules exploiting necessary conditions for feasibility. Informally speaking, the edge finder identifies pairs (Ω, i) such that task i must complete after (resp. start before) all tasks in the set Ω in a feasible schedule, and updates the earliest release date (resp. the latest due date) of i accordingly. An energetic algorithm considers an interval $[t_1, t_2]$ and a task i, approximates the energy consumed in $[t_1, t_2]$ by other tasks, and determines whether task i must end (resp. start) after t_2 (resp. before t_1), in which case its release date (resp. its due date) is updated accordingly.

Complexity results in constraint-based scheduling have focused on a single application of these pruning operators. In particular, for disjunctive scheduling, a single application of the edge finder for the release dates (resp. the due dates) takes $O(n \log n)$ in the worst case, where n is the number of tasks [CP94, VBv04], while it takes $O(n^3)$ time for the cumulative scheduling [MVH05, NA96]. For energetic pruning, the situation is a bit more complex since the complexity depends on the energy function. For instance, the partially elastic pruning operator runs in $O(n^2 \log n)$ time in the worst case [BLPN01]. Unfortunately, these pruning operators are never used in isolation and hence these analyses only present a very partial picture of

their computational complexity. Moreover, this limitation is further exacerbated by the fact that these pruning operators are not idempotent: applying them multiple times may tighten the releases and due dates further. As a consequence, available analyses leave a significant gap in our understanding of these pruning operators and do not provide a sound basis for algorithmic comparison. In fact, based on existing complexity results, it is tempting to conclude that edge-finding algorithms for cumulative scheduling are dominated by energetic pruning, since they produce weaker tightenings at a similar or higher cost.

This paper addresses this limitation and studies, for the first time, the complexity of propagation algorithms for disjunctive and cumulative resources. It contains three main results that may be summarized as follows:

- For disjunctive resources, the fixpoint of the edge finder for both release and due dates can be reached in strongly polynomial time.
- For the cumulative resources, the fixpoint can be reached in strongly polynomial time for updating the release dates or the due dates but not both.
- The fixpoint of "reasonable" energetic operators cannot be reached in strongly polynomial time, even for disjunctive scheduling and even when only the release dates or the due dates are considered.

These results offer a fundamentally different picture of the computational complexity of these pruning operators. In particular, they identify fundamental differences in efficiency when these operators are iterated in fixpoint algorithms. The proofs also indicate some "flaws" in existing definitions of energetic operators; they also suggest some directions in order to address their pathologic behaviors fixpoint computations, especially for cumulative scheduling.

The rest of the paper is organized as follows. Section 2 reviews the technical background behind this paper. Section 3 introduces the concept of propagation patterns to formalize constraint propagation algorithms, which may differ in the order they apply the pruning operators. Section 4 proves some fundamental results on propagation patterns. In particular, it shows that one can essentially focus on a specific class of propagation patterns to derive the complexity results. Section 5 presents the complexity results for disjunctive resources, while Section 6 presents those for cumulative results. Section 7 concludes the paper and presents the open issues. The proofs not given in the paper are in the appendices.

2 Background

This section reviews the main concepts used in the paper. It covers one-resource problems typically arising in constraint-based schedulers, the concept of a pruning operator, the edge finder, and energetic reasoning. Only the concepts relevant to

this paper are presented and readers may consult [BLPN01, Bru95] for excellent references on this topic. Note that some of our definitions are more abstract on order to make the proofs more generic.

2.1 One-Resource Problems

This section defines the one-resource problems tackled by pruning operator based on edge finding and energetic reasoning.

Definition 1 (Cumulative Resource Problems) *A cumulative resource problem (CRP) is a tuple (C, T, p, c, r, d) where*

- *$C \in \mathbb{N}^*$ is the capacity of the resource;*
- *T is a set of $|T| = n$ tasks;*
- *$p_i \in \mathbb{N}$ $(i \in T)$ is the processing time of task i;*
- *$c_i \in \mathbb{N}$ $(i \in T)$ is the capacity requirement of task i;*
- *$r_i \in \mathbb{Z}$ $(i \in T)$ is the release date of task i;*
- *$d_i \in \mathbb{Z}$ $(i \in T)$ is the due date of task i.*

A solution to a CRP $\mathcal{P} = (C, T, p, c, r, d)$ is an assignment of starting dates $s_i \in \mathbb{Z}$ to each task $i \in T$ such that

$$\forall i \in T : r_i \leq s_i \leq s_i + p_i \leq d_i$$

and

$$\forall t : \sum_{\substack{i \in T \\ s_i \leq t < s_i + p_i}} c_i \leq C.$$

The set of solutions to a CRP $\mathcal{P}$ is denoted by $\mathrm{sol}(\mathcal{P})$.

In the following, we use the terms *solution* and *schedule* interchangeably. Disjunctive resource problems, that have a single unit of capacity at all times, play an important role in practice.

Definition 2 (Disjunctive Resource Problems) *A disjunctive resource problem (DRP) is a cumulative resource problem in which $C = 1$ and $\forall i \in T, c_i = 1$.*

Although they only consider a single resource and are generally parts of more complex problems, CRPs and DRPs are already NP-complete. Note also the integrality of the starting dates in these problems, which is both natural in practice and fundamental for the results of this paper.

2.2 Problem Tightenings and Pruning Operators

The fundamental operation of constraint-based schedulers consists of tightening the release and the due dates of the tasks without removing any solution. It is convenient to formalize this process using the concept of problem tightening.

Definition 3 (Problem Tightening) *Let $\mathcal{P}$ be a CRP (C, T, p, c, r, d). A CRP $\mathcal{P}' = (C, T, p, c, r', d')$ is a tightening of $\mathcal{P}$, denoted by $\mathcal{P}' \subseteq \mathcal{P}$, if it satisfies the following two conditions:*

(soundness) $\mathrm{sol}(\mathcal{P}) = \mathrm{sol}(\mathcal{P}')$*;*

(contractance) $\forall i \in T, r'_i \geq r_i \ \wedge \ d'_i \leq d_i$*.*

The intuition here is that a problem tightening reduces the search space, i.e., the set of possible values for the starting dates, without removing any solution. Ideally one would like to obtain the strongest possible tightening but this cannot be solved in polynomial time (unless $P = NP$), since the CRP (resp. DRP) is NP-complete. This is precisely the reason why several pruning operators have been proposed: these operators achieve different tightenings at different computation costs. It is the objective of this paper to offer a better understanding of their computational complexity.

Definition 4 (Pruning Operator) *A pruning operator ψ is a function which, given a CRP $\mathcal{P}$, returns a tightening of $\mathcal{P}$ and is monotone,*[1] *i.e.,*

$$\forall \mathcal{P}_1, \mathcal{P}_2, \quad \mathcal{P}_1 \subseteq \mathcal{P}_2 \implies \psi(\mathcal{P}_1) \subseteq \psi(\mathcal{P}_2).$$

A pruning operator ψ is idempotent if $\psi \circ \psi = \psi$.

Most papers focus exclusively on tightening the release dates, since the treatment is symmetric for due dates. The next definition captures this formally.

Definition 5 *Let ψ be a pruning operator and R be the time-reversal function $R : (C, T, p, c, r, d) \mapsto (C, T, p, c, -d, -r)$. The function $\psi^R = R \circ \psi \circ R$ is a pruning operator called the symmetric version of ψ.*

All the operators discussed in this paper deal with release dates or due dates but not both. We use the suffixes -R and -D for naming operators focusing on release and due dates respectively.

[1] Strictly speaking, pruning operator needs not be monotone, although this is natural. Monotonicity is important for several results in this paper and all the pruning operators considered here are monotone.

2.3 Notations

Since problem tightenings only affect the release and due dates of the tasks, we abuse notations and denote CRPs (and DRPs) by pairs (r, d) assuming that the remaining data, i.e., the tuple (C, T, p, c), is fixed. We define the energy e_i of a task $i \in T$ as $e_i = c_i p_i$. We also lift the above concepts to sets of tasks, i.e.,

$$\begin{aligned} r_\Omega &= \min_{j\in\Omega} r_j \\ d_\Omega &= \max_{j\in\Omega} d_j \\ p_\Omega &= \sum_{j\in\Omega} p_j \\ e_\Omega &= \sum_{j\in\Omega} e_j \end{aligned}$$

where Ω is a set of tasks. By convention, $r_\emptyset = \infty$, $d_\emptyset = -\infty$, $p_\emptyset = 0$, and $e_\emptyset = 0$. We also use $\mathcal{P}_\emptyset$ to denote a canonical infeasible problem, e.g., a problem with $r_i = +\infty$ and $d_i = -\infty$ for each task i. Note that any pruning operator ψ satisfy $\psi(\mathcal{P}_\emptyset) = \mathcal{P}_\emptyset$.

2.4 Edge Finding

Edge-finding operators are a cornerstone of constraint-based schedulers. We start by specifying the disjunctive case, whose intuition is easier to convey.

Definition 6 (Disjunctive Edge Finder (Release Dates)) *Let $\mathcal{P} = (r, d)$ be a DRP. The edge finder EF-R for $\mathcal{P}$ returns $\mathcal{P}_\emptyset$ if*

$$\exists\ \Omega \subseteq T :\ r_\Omega + p_\Omega > d_\Omega.$$

Otherwise, it returns the DRP $\mathcal{P}' = (\overline{LB}, d)$ such that

$$\overline{LB}_i = \max(r_i, LB_i)$$

and

$$LB_i = \max_{\substack{\Omega \subseteq T \\ i \notin \Omega \\ \alpha(\Omega, i)}} \ \max_{\Theta \subseteq \Omega} \ r_\Theta + p_\Theta$$

where

$$\alpha(\Omega, i) \iff \left(d_\Omega - r_{\Omega\cup\{i\}} < p_{\Omega\cup\{i\}}\right).$$

Its intuition underlying the edge finder can be described informally as follows. First, if there exists a set $\Omega \subseteq T$ such that

$$r_\Omega + p_\Omega > d_\Omega$$

then the tasks in Ω cannot be scheduled in the interval $[r_\Omega, d_\Omega($ and hence the DRP has no solution. Second, the general case considers pairs (Ω, i) where Ω is a set of

tasks and i is a specific task. The condition

$$\alpha(\Omega, i)$$

holds when there is not enough time between $r_{\Omega\cup\{i\}}$ and d_Ω to process all the tasks in $\Omega \cup \{i\}$. As a consequence, task i must start after the completion of all tasks of Ω in all solutions. The expression

$$\max_{\Theta \subseteq \Omega} \quad r_\Theta + p_\Theta$$

provides a lower bound on the completion time of the tasks in Ω and thus a new candidate release date for task i. More generally, the candidate release date LB_i of task i is obtained by computing this lower bound for all subsets Ω satisfying $\alpha(\Omega, i)$.

The edge finder is a pruning operator [CP94] and it can be computed in $O(n \log n)$ time [CP94, VBv04]. (A similar result also holds for due dates using Definition 5.) Unfortunately, the edge finder is not idempotent and it must be iterated until no more updates take place. It is precisely the goal of this paper to analyze how many iterations are necessary to reach such a state.

Nuijten [NA96] generalized the edge finder to the cumulative case.

Definition 7 (Cumulative Edge Finder (Release Dates)) *Let $\mathcal{P} = (r, d)$ be a CRP. The edge finder EF-R for $\mathcal{P}$ returns $\mathcal{P}_\emptyset$ if*

$$\exists\ \Omega \subseteq T : e_\Omega > C(d_\Omega - r_\Omega) \quad \vee \quad \exists i \in T : r_i + p_i > d_i.$$

Otherwise, it returns the CRP $\mathcal{P}' = (\overline{LB}, d)$ such that

$$\overline{LB}_i = \max(r_i, LB_i)$$

and

$$LB_i = \max_{\substack{\Omega \subseteq T \\ i \notin \Omega \\ \alpha(\Omega, i)}} \ \max_{\substack{\Theta \subseteq \Omega \\ \mathrm{rest}(\Theta, c_i) > 0}} \ r_\Theta + \left\lceil \frac{1}{c_i} \mathrm{rest}(\Theta, c_i) \right\rceil$$

where

$$\begin{cases} \alpha(\Omega, i) \iff \left(C(d_\Omega - r_{\Omega\cup\{i\}}) < e_{\Omega\cup\{i\}}\right). \\ \mathrm{rest}(\Theta, c_i) = \begin{cases} e_\Theta - (C - c_i)(d_\Theta - r_\Theta) & \text{if } \Theta \neq \emptyset; \\ 0 & \text{otherwise.} \end{cases} \end{cases}$$

Here is the intuition underlying the cumulative edge finder. The basic idea is to reason about the so-called energy of a task, i.e., the product $p_i \times c_i$ of the processing time and the capacity requirement of task i. The cumulative edge finder detects infeasibility if

$$\exists\ \Omega \subseteq T : e_\Omega > C(d_\Omega - r_\Omega),$$

i.e., if there is a subset of tasks Ω whose energy is greater than the capacity of the resource in the interval $[r_\Omega, d_\Omega($. It also detects infeasibility if there is a task i such that $r_i + p_i > d_i$.

The general case for the cumulative edge finder is similar in spirit to the disjunctive case, but it reasons about energies instead of processing times which mostly complicates the lower bound computation. The generalization of condition α to the cumulative case is natural: It considers a set of tasks Ω and a task i and it holds if

$$e_{\Omega\cup\{i\}} > C(d_\Omega - r_{\Omega\cup\{i\}}),$$

i.e., if the energy requires by $\Omega\cup\{i\}$ is greater than the capacity in interval $[r_{\Omega\cup\{i\}}, d_\Omega($.

However, the conclusion that can be inferred when $\alpha(\Omega, i)$ holds is weaker in the cumulative case. Indeed, task i does not necessarily start after all tasks in Ω; we can only infer than task i must end after d_Ω. The computation of the lower bound on r_i is thus quite different from the disjunctive case and deserves some additional explanation.

The lower bound is computed by reasoning, once again, about energies. We know that task i is necessarily running from s_i to d_Ω in each solution s, taking c_i capacity units from the resource. To determine a lower bound on s_i, the edge finder considers each set $\Theta \subseteq \Omega$, which requires an energy e_Θ in interval $[r_\Theta, d_\Theta($. The edge finder first removes from e_Θ the energy that can be scheduled in $[r_\Theta, d_\Theta($ without interfering with task i, i.e.,

$$(C - c_i)(d_\Theta - r_\Theta).$$

This leaves in a slice of capacity c_i to schedule task i and the remaining energy

$$\text{rest}(\Theta, c_i) = e_\Theta - (C - c_i)(d_\Theta - r_\Theta).$$

If the remaining energy is smaller or equal to zero, the tasks in Θ do not constraint task i and no new release date results. Otherwise, a lower bound on the release date of tasks i is obtained by determining how much of the slice of capacity c_i is taken by the remaining energy starting from r_Θ, giving

$$r_\Theta + \left\lceil \frac{1}{c_i} \text{rest}(\Theta, c_i) \right\rceil.$$

The fastest correct algorithm for the cumulative edge finder runs in $O(n^2 \left|\{c_i, i \in T\}\right|)$ [MVH05]. Note that,once again, the cumulative edge-finder is not idempotent.

2.5 Energy-Based Operators

A variety of elegant and interesting pruning operators for cumulative resource problem are based on energetic reasoning. Given a task i, these operators rely on a lower bound of the energy consumed in an interval $[t_1, t_2]$ by the tasks in $T \setminus \{i\}$ in all

feasible schedules. Once such a lower bound is available, the energetic operators determine whether task i must finish after time t_2 (resp. start before time t_1) and update its release date (resp. due date) accordingly.

Several lower bounds for disjunctive and cumulative resources have been proposed and achieve different tradeoffs in quality and computational complexity. Our result apply generically to all "reasonable" energetic operators, a concept defined below. To formalize energetic operators, we first specify the resource consumption of a task i in an interval $[t_1, t_2]$ for a *specific* schedule s.

Definition 8 (Resource Consumption) *Let $\mathcal{P} = (r, d)$ be a CRP, i be a task, and s be a schedule of $\mathcal{P}$. The resource consumption of task i over $[t_1, t_2]$ in s is*

$$X(s, i, t_1, t_2) = c_i\Big(\min(t_2, s_i + p_i) - \max(t_1, s_i)\Big).$$

In the definition, the expression

$$\min(t_2, s_i + p_i) - \max(t_1, s_i)$$

represents the processing time of task i spent within the interval $[t_1, t_2]$. This definition can be lifted to sets of tasks, i.e.,

$$X(s, \Omega, t_1, t_2) = \sum_{i \in \Omega} X(s, i, t_1, t_2).$$

Note that a schedule s satisfies the cumulative resource constraint if and only if

$$\forall t_1 \leq t_2 : X(s, T, t_1, t_2) \leq C(t_2 - t_1).$$

The next definition, energy functions, is fundamental in energic reasoning. An energy function receives a CRP $\mathcal{P}$, a task i, and an interval $[t_1, t_2]$ as inputs, and returns a lower bound on the resource consumption of task i in $[t_1, t_2]$ *for all the feasible schedules of $\mathcal{P}$*.

Definition 9 (Energy Function) *Let $\mathcal{P}$ be a CRP. An energy function for $\mathcal{P}$ is a function $W : T \times \mathbb{Z} \times \mathbb{Z} \to \mathbb{N}$ satisfying*

$$\forall s \in \mathrm{sol}(\mathcal{P}), \forall i \in T, \forall t_1 \leq t_2 : \quad W(i, t_1, t_2) \leq X(s, i, t_1, t_2).$$

Informally speaking, an energy function provides a lower bound of the resource consumption of a task in an interval for all feasible schedules of a CRP simultaneously. Once again, this concept can be lifted to sets of tasks, i.e.,

$$W(\Omega, t_1, t_2) = \sum_{i \in \Omega} W(i, t_1, t_2).$$

Our results apply generically to all "reasonable" energy function. Informally speaking, an energy function $W(i, t_1, t_2)$ is reasonable if it returns e_i when task i *must* execute in $[t_1, t_2]$.

Definition 10 (Reasonable Energy Function) *Consider the energy function*

$$W_{Plain}(i, t_1, t_2) = \begin{cases} e_i & \text{if } r_i \geq t_1 \text{ and } d_i \leq t_2 \\ 0 & \text{otherwise} \end{cases}$$

for a CRP $\mathcal{P}$. An energy function W for $\mathcal{P}$ is reasonable if

$$\forall i \in T, \forall t_1 \leq t_2 : W(i, t_1, t_2) \geq W_{Plain}(i, t_1, t_2).$$

As mentioned earlier, a variety of energy functions are available: they include W_{FE} (Fully Elastic), W_{PE} (Partially Elastic) and W_{Sh} (Left-Shift-Right-Shift). Moreover, as shown in [BLPN01], they satisfy

$$W_{Sh} \geq W_{PE} \geq W_{FE} \geq W_{Plain}.$$

We are now in a position to specify the pruning operator associated with an energy function.

Definition 11 (Energetic Pruning for Release Dates) *Let $\mathcal{P} = (r, d)$ be a CRP and W be an energy function. The energetic operator W-R for $\mathcal{P}$ returns $\mathcal{P}_\emptyset$ if*

$$\exists\ t_1 \leq t_2 : W(T, t_1, t_2) > C(t_2 - t_1) \quad \vee \quad \exists i \in T : r_i + p_i > d_i.$$

Otherwise, it returns the CRP $\mathcal{P}' = (\overline{ELB}, d)$ such that

$$\overline{ELB}_i = \max(r_i, ELB_i)$$

and

$$ELB_i = \max_{\substack{t_1 < t_2 \\ \Delta(i, t_1, t_2) > 0}} t_2 + \left\lceil \frac{1}{c_i} \Delta(i, t_1, t_2) \right\rceil - p_i$$

with

$$\Delta\big(i, t_1, t_2\big) = W\big(T \setminus \{i\}, t_1, t_2\big) \ + \ c_i p_i^+(t_1) \ - \ C(t_2 - t_1)$$

and

$$p_i^+(t_1) = \max(0, p_i - \max(0, t_1 - r_i))$$

Let us go over the definition to convey the intuition of energetic operators. The condition

$$\Delta\big(i, t_1, t_2\big) > 0$$

holds whenever task i must finish after time t_2. It is expressed in terms of the energy of the tasks $T \setminus \{i\}$ in $[t_1, t_2]$, i.e.,

$$W\big(T \setminus \{i\}, t_1, t_2\big)$$

and the energy that task i must spend after t_1, i.e.,

$$c_i p_i^+(t_1).$$

In this last expression, $p_i^+(t_1)$ represents a lower bound on the processing time of task i after t_1. If

$$W\big(T \setminus \{i\}, t_1, t_2\big) \; + \; c_i p_i^+(t_1) \quad > \quad C(t_2 - t_1)$$

there is not enough energy to schedule task i before t_2, so i must necessarily finish after. Moreover, $\Delta(i, t_1, t_2)$ represents the energy of task i that must be spent after t_2. A lower bound on the earliest finishing date of task i is thus given by

$$t_2 + \left\lceil \frac{1}{c_i} \Delta(i, t_1, t_2) \right\rceil$$

and a lower bound on its release date is obtained by subtracting its processing time:

$$t_2 + \left\lceil \frac{1}{c_i} \Delta(i, t_1, t_2) \right\rceil - p_i.$$

The energetic operator W-R applies this reasoning for all tasks and all intervals $[t_1, t_2]$. A symmetric operator W-D can be defined for due dates. These operators are not idempotent in general.

Note that the stronger the energy function, the stronger the condition, and the stronger the lower bound on the release dates. As a consequence, we have

$$W_{Sh}\text{-R}(\mathcal{P}) \subseteq W_{PE}\text{-R}(\mathcal{P}) \subseteq W_{FE}\text{-R}(\mathcal{P}).$$

There are however some interesting open computational issues for energetic operators. Indeed, although W_{PE}-R$(\mathcal{P})$ can be computed in polynomial time, no polynomial algorithm is known for W_{Sh}-R$(\mathcal{P})$. Also interesting is the fact that the fixpoint of operator W_{PE}-R is included in the fixpoint of EF-R. Since W_{PE}-R$(\mathcal{P})$ can be computed in $O(n^2 \log |\{c_i, i \in T\}|)$, it is tempting to conclude that W_{PE}-R$(\mathcal{P})$ should always be prefered to EF-R$(\mathcal{P})$. *One of the corollaries of this paper is that this conclusion should be revised, because the fact that one iteration of W_{PE}-R is faster than one of EF-R does not imply that the fixpoint of W_{PE}-R can be computed faster than the fixpoint of EF-R.*

3 Propagation Algorithms and Propagation Patterns

The main goal of this paper is to study the computational complexity of applying a set of pruning operators

$$\{\psi_1, \ldots, \psi_k\}$$

to a fixpoint, i.e., until the tightenings produce a CRP $\mathcal{P}$ satisfying

$$\mathcal{P} = \psi_1(\mathcal{P}) \ \wedge \ \ldots \ \wedge \ \mathcal{P} = \psi_k(\mathcal{P}).$$

Since the pruning operators are monotone and since the release and due dates are integers, the resulting fixpoint is independent of the application order and is reached in finite time. When $k > 1$, many strategies can be applied to reach the fixpoint, but they may not be equivalent from a complexity standpoint. Consider, for instance, the case of $k = 2$ and the operators EF-R and EF-D. One strategy to reach the fixpoint consists of applying EF-R, then EF-D, and to iterate this process until an iteration does not tighten the release or the due dates. Another strategy applies EF-R to a fixpoint, then EF-D to a fixpoint, and iterates the process until a global fixpoint is reached. A third strategy would apply EF-R to a fixpoint, then EF-D once, and iterate the process. Are these strategies equivalent from a computational standpoint? If not, which one should be preferred?

This section addresses this issue through the concept of propagation patterns which specify the strategy to apply in order to reach the fixpoint of a set of operators. Propagation patterns are expressed in terms of pruning operators, sequential composition, and the Kleen fixpoint operator.[2] For instance, $(\text{EF-R})^\star$ denotes the fixpoint of EF-R, $(\text{EF-R} \cdot \text{EF-D})^\star$ the pattern that applies EF-R and EF-D in sequence until a fixpoint is reached, and $((\text{EF-R})^\star \cdot (\text{EF-D})^\star)^\star$ the pattern that applies $(\text{EF-R})^\star$ and $(\text{EF-D})^\star$ in sequence until a fixpoint is reached. We now formalize these concepts.

Definition 12 (Propagation and Fixpoint Patterns) *Let $\Psi = \{\psi_1, \ldots, \psi_k\}$ be a set of pruning operators. A propagation pattern is a string over the alphabet $\Psi \cup \{(, \cdot,), ^\star\}$ that contains at least one occurrence of ψ_i ($1 \leq i \leq k$) and is derived from the grammar*

$$\langle \mathsf{pattern} \rangle \quad ::= \quad \psi_i \mid (\langle \mathsf{pattern} \rangle)^\star \mid \langle \mathsf{pattern} \rangle \cdot \langle \mathsf{pattern} \rangle$$

A fixpoint pattern is a pattern of the form $(F)^\star$, where F is a propagation patterns.

The semantics of propagation and fixpoint patterns is given by algorithm `Propagate` which returns the sequence of CRPs produced by applying the pruning operators. (In the algorithm, $S_1 :: S_2$ denotes the concatenation of two sequences S_1 and S_2). The algorithm is defined inductively on the structure of propogation patterns. The

[2] Constraint programming systems and constraint-based schedulers may also introduce some randomization in the propagation algorithms. There is no difficulty in generalizing the results presented here to address this additional functionality.

basic case of pruning operators is shown in Lines 2–3; it returns a sequence of one CRP obtained by applying an operator ψ on $\mathcal{P}$. Lines 4–6 define the sequential composition of patterns F_1 and F_2. The output sequence is the concatenation of the sequence $\langle \mathcal{P}_1, \ldots, \mathcal{P}_q \rangle$ obtained by applying F_1 on $\mathcal{P}$ with the sequence resulting from the application of F_2 on $\mathcal{P}_q$. Finally, lines 7–12 implement the fixpoint of pattern F_1. Line 8 applies F_1 on $\mathcal{P}$ to obtain the sequence $\langle \mathcal{P}_1, \ldots, \mathcal{P}_q \rangle$. If $\mathcal{P}_q = \mathcal{P}$, the algorithm has reached a fixpoint and it returns the sequence $\langle \mathcal{P}_1, \ldots, \mathcal{P}_q \rangle$. Otherwise, the algorithm applies the pattern F_1 recursively on $\mathcal{P}_q$ and returns the concatenation of the first and subsequent applications of F_1.

Function `Propagate(`$\mathcal{P}$`, `F`)`

Precondition: $\mathcal{P}$ is a CRP and F is a fixpoint pattern over $\{\psi_1, \ldots, \psi_k\}$.
Output: A sequence $\langle \mathcal{P}_1, \ldots, \mathcal{P}_Q \rangle$ of CRPs.
Postcondition: $\mathrm{sol}(\mathcal{P}_Q) = \mathrm{sol}(\mathcal{P})$ and $\psi_i(\mathcal{P}_Q) = \mathcal{P}_Q \;\; (1 \leq i \leq k)$.

```
1  match F with
2  | case ψ
3  |   return ⟨ψ(P)⟩
4  | case F1 · F2
5  |   let ⟨P1, ..., Pq⟩ = Propagate(P,F1) in
6  |     return ⟨P1, ..., Pq⟩::Propagate(Pq,F2)
7  | case (F1)*
8  |   let ⟨P1, ..., Pq⟩ = Propagate(P,F1) in
9  |     if Pq = P then
10 |       return ⟨P1, ..., Pq⟩
11 |     else
12 |       return ⟨P1, ..., Pq⟩::Propagate(Pq,F1)
```

In practice, only the last CRP of the sequence is of interest. However, representing the sequence explicitly is important for the complexity proofs. Indeed, the results in this paper are all concerned with bounding the length of the output sequence of `Propagate` for different patterns and pruning operators. We now define the complexity measures used in this paper. These measures capture strong polynomiality without the need to refer to models of computation such as arithmetic machines.

Definition 13 (Complexity of Propagation Patterns) *The complexity of a polynomial pattern F is the function $C_F : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ defined as follows: $C_F(n, h)$ is the maximal length of the sequence returned by* `Propagate`$(F, \mathcal{P})$ *for a CRP (resp. DRP) $\mathcal{P}$ with at most n tasks and whose release and due dates satisfy*

$$\forall i \in T,\ r_i \geq -h \ \wedge \ d_i \leq h.$$

Our main interest in this paper is to determine whether the complexity of a propagation pattern depends on the horizon h. This justifies the next definition.

Definition 14 (Strong Polynomiality of Propagation Patterns) *A propagation pattern F is polynomial if its complexity function $C_F(n, h)$ is bounded by a polynomial in n and $\log h$. It is horizon-independent if $C_F(n, h)$ is bounded by a function of n. Finally, a propagation pattern F is strongly polynomial if $C_F(n, h)$ is bounded by a polynomial in n.*

Observe that the complexity of a propagation pattern is independent from the complexity of its pruning operators. This is intentional since they are independent: our results do not depend on the actual implementations of the pruning operators, whose complexity is sometimes open. However, it is easy to derive the overall complexity of the propagation from the definition. In particular, if a propagation pattern F over $\Psi = \{\psi_1, \ldots, \psi_k\}$ is strongly polynomial and each pruning operator ψ_i is strongly polynomial (i.e., its runtime is bounded by a polynomial in n on an arithmetic machine), the propagation of F is guaranteed to be strongly polynomial. Similarly, if each operator is polynomial (i.e., its runtime is bounded by a polynomial in n and $\log h$) and that the pattern is polynomial, then the propagation process runs in polynomial time. On the other hand, if F is not strongly polynomial, then it makes little sense to improve the complexity of the pruning operators, since the propagation of F will not be strongly polynomial. Note also that the (disjunctive and cumulative) edge finders and the partial elastic energetic operator are strongly polynomial.

4 Fundamental Properties of Propagation Patterns

This section studies the properties of propagation patterns. Its main result is to show that, when the goal is to prove strong polynomiality, the complexity analysis can focus on a simple class of propagation patterns that are, in some sense, "optimal". The section also proves that the number of "useless" applications of pruning operators is essentially amortized by their "useful" counterparts, i.e., the applications tightening the release or due dates. Our first result is part of the folklore of constraint programming and constraint-based scheduling.

Theorem 1 *Let F_1 and F_2 be two fixpoint patterns over $\Psi = \{\psi_1, \ldots, \psi_k\}$ and $\mathcal{P}$ be a CRP. Then,* $\textsc{last}(\texttt{Propagate}(\mathcal{P}, F_1)) = \textsc{last}(\texttt{Propagate}(\mathcal{P}, F_2))$*, where* $\textsc{last}(S)$ *denotes the last element of a non-empty sequence.*

Proof See appendix A. ■

Our next theorem identifies a specific class of fixpoint patterns with strong computational properties. More precisely, it shows that the propagation pattern $(\psi_1 \cdot \ldots \cdot \psi_p)^\star$ returns sequences that are at most p times longer than the sequences returned by any other propagation pattern over $\{\psi_1, \ldots, \psi_p\}$.

Theorem 2 *Let F be a fixpoint pattern over $\{\psi_1, \ldots, \psi_p\}$ and consider the pattern $O = (\psi_1 \cdot \ldots \cdot \psi_p)^\star$. If $C_F(n,h)$ is $O(f(n,h))$, then $C_O(n,h)$ is $O(f(n,h))$ too.*

Proof Consider the sequence $\langle \mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_q \rangle$, where $\mathcal{P}_0 = \mathcal{P}$ and $\langle \mathcal{P}_1, \ldots, \mathcal{P}_q \rangle$ is the sequence returned by `Propagate`$(\mathcal{P}, F)$. Define

$$PO_j = (\psi_p \circ \cdots \circ \psi_1)^j (\mathcal{P}).$$

We first prove by induction that $\forall j : PO_j \subseteq \mathcal{P}_j$. This holds for $j = 0$. Assume now that $PO_j \subseteq \mathcal{P}_j$. There exists $i \in \{1, \ldots, p\}$ such that $\mathcal{P}_{j+1} = \psi_i(\mathcal{P}_j)$. Since

$$PO_{j+1} = (\psi_p \circ \cdots \circ \psi_{i+1}) \circ \psi_i \circ (\psi_{i-1} \circ \cdots \circ \psi_1)(PO_j)$$

and since the operators are contractant and monotone, it follows that $PO_{j+1} \subseteq \mathcal{P}_{j+1}$.

Assume now that `Propagate`$(\mathcal{P}, O)$ returns a sequence $\mathcal{P}'$ of length Q and suppose that $Q > pq$. Denote by $\mathcal{P}'_{pq}$ the element in position pq in $\mathcal{P}'$. By definition of O, we have that $\mathcal{P}'_{pq} = PO_q \subseteq \mathcal{P}_q$. Moreover, by Theorem 1, $\mathcal{P}_q \subseteq \mathcal{P}'_Q$. Since $\mathcal{P}'_Q \subseteq \mathcal{P}'_{pq}$, we have $\mathcal{P}_q \subseteq \mathcal{P}'_{pq}$ and thus $\mathcal{P}_q = \mathcal{P}'_{pq}$. Hence, `Propagate`$(\mathcal{P}, O)$ returns a sequence of length at most $p(q+1)$, since an additional iteration with p operator applications detects that the fixpoint has been reached. ■

Theorem 2 allows us to restrict attention to a single pattern when proving that the propagation of a set of pruning operators is not strongly polynomial. Indeed, if $(\psi_1 \cdot \ldots \cdot \psi_p)^\star$ is not strongly polynomial, no other propagation pattern will be. The proof also shows that a specific polynomial pattern can speed up the naive pattern by a factor p at most. Note also that a common propagation pattern is

$$\left(\left(\ldots((\psi_1)^\star \cdot (\psi_2)^\star)^\star \ldots \cdot (\psi_{p-1})^\star\right)^\star \cdot (\psi_p)^\star\right)^\star$$

which is thought to eliminate many useless operator applications after a fixpoint is reached. Theorem 2 suggests that it may be better to apply the "naive" pattern $(\psi_1 \ldots \psi_p)^\star$ which may avoid many iterations with little pruning.

The next result links the tightenings of the release and due dates using the time reversal function from Definition 5.

Theorem 3 *Let F be a pattern and let F^R be its time-symmetric version, i.e., the pattern obtained from F by replacing each ψ in F by its time-symmetric version $R \circ \psi \circ R$. Consider a CRP $\mathcal{P}$ and let $\langle \mathcal{P}_1, \ldots, P_Q \rangle =$ `Propagate`$(\mathcal{P}, F)$ and $\langle \mathcal{P}'_1, \ldots, P'_{Q'} \rangle =$ `Propagate`$(R(\mathcal{P}), F^R)$. We have*

$$Q = Q' \ \wedge \ \forall\, 0 \leq q \leq Q : \ \mathcal{P}'_q = R(\mathcal{P}_q).$$

Proof (Sketch) By induction on the language of propagation patterns. ■

Corollary 1 *A pattern F and its time-symmetric version F^R have the same complexity.*

Proof Direct consequence of Theorem 3. ■

The next theorem is important: it shows that the number of operator applications is bounded by a linear function in the number of "useful" applications, i.e., those applications that actually update the release or the due dates of tasks. This theorem will be instrumental in showing that, in some cases, all patterns over a given set of operators are equivalent from a complexity standpoint.

Theorem 4 *Let F be a propagation pattern. There exist 2 constants a and b satisfying $0 < a \leq 1$ and $b \geq 1$ such that, for any CRP $\mathcal{P}_0$, we have*

$$\left|\left\{0 \leq q < Q \;\middle|\; \mathcal{P}_{q+1} \neq \mathcal{P}_q\right\}\right| \quad \geq \quad aQ - b$$

where

$$\langle \mathcal{P}_1, \ldots, P_Q \rangle = \texttt{Propagate}(\mathcal{P}_0, F).$$

Proof The proof is by induction on the structure of the propagation pattern and is given in Appendix B. ■

Observe that the proof of Theorem 4 does not guarantee that the choices of (a, b) are tight. It would be interesting to obtain such tight bounds since this would provide a basis for comparing patterns with similar numbers of "useful" calls.

Corollary 2 *If F is a propagation pattern, $C_F(n, h)$ is $O(nh)$.*

Proof Let $\langle \mathcal{P}_1, \ldots, \mathcal{P}_Q \rangle = \texttt{Propagate}(\mathcal{P}_0, F)$. By monotonicy, every pruning operator in F satisfies $\psi(\mathcal{P}_\emptyset) = \mathcal{P}_\emptyset$. Hence, since the release and due dates are all in $[-h, h]$, the number of integer q such that $\mathcal{P}_{q+1} \neq \mathcal{P}_q$ is less than $2nh$. By Theorem 4, there exists $0 < a \leq 1$ and b such that $2nh > aQ - b$. Thus Q is $O(nh)$. ■

5 Disjunctive Scheduling

We now return to resource propagation algorithms and first consider the case of disjunctive scheduling. We show that fixpoint patterns over EF-R and EF-D are strongly polynomial and that energetic operator are not.

Our first result bounds the number of useful applications of EF-R.

Theorem 5 *Let Ψ be a set of pruning operators containing EF-R such that any other operator in Ψ do not modify release dates, i.e.,*

$$\forall\ \psi \in \Psi \setminus \{\textit{EF-R}\}, \forall\ \mathcal{P} = (r, d) :\ \exists\ d' \in \mathbb{Z}^n :\ \psi(\mathcal{P}) = (r, d').$$

Let F be a propagation pattern on Ψ, $\mathcal{P}_0$ be a DRP, and

$$\langle \mathcal{P}_1, \ldots, \mathcal{P}_Q \rangle = \texttt{Propagate}(\mathcal{P}_0, F).$$

Denote by

$$\langle \psi_0, \ldots, \psi_{Q-1} \rangle$$

the sequence of pruning operators applied to produce $\langle \mathcal{P}_1, \ldots, \mathcal{P}_Q \rangle$, i.e.,

$$\psi_q(\mathcal{P}_q) = \mathcal{P}_{q+1}.$$

We have that

$$\left| \{0 \leq q < Q \mid \psi_q = \textit{EF-R} \wedge \mathcal{P}_{q+1} \neq \mathcal{P}_q\} \right| \quad \leq \quad n^3.$$

Proof (Sketch) The full proof is in Appendix D.[3] It consists of identifying the reasons for an update at a given step. There are two such reasons that are non-exclusive:

1. either a pair (Ω, i) now satisfies $\alpha(\Omega, i)$, which was not the case before;
2. or the release date of a task j has been increased, changing the release date of a task i satisfying $\alpha(\Omega, i)$ for a set Ω with $j \in \Omega$.

The first reason can occur at most n^2 times, since the discovery of a new pair (Ω, i) implies that Ω precedes task i in all feasible schedules and the precedence graph cannot have more than n^2 arcs.

The second reason considers only subsequences of updates involving operator EF-R, since other operators cannot tighten the release dates. We show that the lengths of these subsequences are bounded by $n - 1$ because the updates take place by increasing order of the due dates. The result follows. ■

We are now in position to show the strong polynomiality of the edge-finder propagation for disjunctive scheduling.

Corollary 3 (The Disjunctive Edge Finder is Strongly Polynomial) *Let F be a propagation pattern over {EF-R, EF-D}. Then $C_F(n, h)$ is $O(n^3)$ for DRPs.*

[3] The proof uses a lemma which is also valid for cumuative scheduling, so that the appendices present the cumulative results first.

Proof Let $\mathcal{P}_0$ be a DRP and

$$\langle \mathcal{P}_1, \ldots, \mathcal{P}_Q \rangle = \texttt{Propagate}(\mathcal{P}_0, F).$$

Define $\mathcal{A} = \{0 \leq q < Q \mid \mathcal{P}_{q+1} \neq \mathcal{P}_q\}$ and partition $\mathcal{A}$ into $\mathcal{A}_R = \mathcal{A} \cap \mathcal{R}$ and $\mathcal{A}_D = \mathcal{A} \setminus \mathcal{R}$, where $\mathcal{R}$ is the set of (EF-R) applications. By Theorem 5, $|\mathcal{A}_R| \leq n^3$ and by Theorems 5 and 3, $|\mathcal{A}_D| \leq n^3$. Hence $|\mathcal{A}| \leq 2n^3$. By Theorem 4, the length Q is $O(|\mathcal{A}|)$ and thus Q is $O(n^3)$. ■

Corollary 3 is a significant improvement over prior work. Indeed, it was generally believed that the edge-finder fixpoint is pseudo-polynomial and can take up to $O(nh)$ applications of the edge-finder operators. Corollary 3 shows that the edge-finder fixpoint is strongly polynomial and thus horizon-independent. Corollary 3 can also be extended to the complexity of maintaining the edge-finder fixpoint in a branch of a search tree, since the number of precedence constraints in such a branch is bounded by n^2. *As a consequence, the constraint propagation in jobshop scheduling algorithms, which combine the edge-finder fixpoint with nondeterministic choices adding precedence constraints in an acyclic manner, is strongly polynomial along a branch of the search tree.* It is an open issue to determine if the $O(n^3)$ bound in Corrolary 3 is tight. We conjecture that it is not, because of the inherent structure of precedence constraints.

Our next result shows that, contrary to edge-finding algorithms, the fixpoint of a single energetic pruning operator is not strongly polynomial.

Theorem 6 *If W is a reasonable energy function, then pattern $(W\text{-}R)^\star$ is not horizon-independent for DRPs.*

Proof Let $p \in \mathbb{N}$ and consider the following instance $\mathcal{P}_0$:

task	r	d	p
a	0	$2p+2$	$p+1$
b	p	$p+1$	1

Define $\mathcal{P}_q$ recursively as

$$\mathcal{P}_{q+1} = W\text{-R}(\mathcal{P}_q) \quad (q \in \mathbb{N})$$

and denote by $\mathcal{P}_q$ the DRP (r^q, d^q) and by W^q the energy function W applied to $\mathcal{P}_q$. We show that, for all $q \leq p+1$,

$$r_a^q = q \qquad r_b^q = p$$
$$d_a^q = 2p+2 \qquad d_b^q = p+1.$$

First note that $(s_a = p+1, s_b = p)$ is a feasible schedule. Hence no infeasibility can be detected and

$$r_b^q = p$$

Moreover, since (W-R) does not modify the due dates,

$$d_a^q = 2p + 1 \quad \wedge \quad d_b^q = p + 1.$$

It remains to show $r_a^q = q$.

The proof is by induction. Clearly the equalities hold for $q = 0$. Now assume that it holds for a given $q \leq p$. As $T \setminus \{a\} = \{b\}$, we only need to consider $W^q(b, t_1, t_2)$. If $t_2 \leq p$ or if $t_1 \geq p + 1$, then $X(s, b, t_1, t_2) = 0$ in any feasible schedule s and

$$W^q(b, t_1, t_2) = 0.$$

If $t_1 \leq p$ and $t_2 \geq p + 1$, then

$$W^q_{Plain}(b, t_1, t_2) = 1.$$

Since

$$W^q_{Plain}(b, t_1, t_2) \leq W^q(b, t_1, t_2) \leq e_b = 1,$$

$W^q(b, t_1, t_2) = 1$. Hence, since W is a reasonable energy function, we have

$$W^q(b, t_1, t_2) = \begin{cases} 1 & \text{if } t_1 \leq p \text{ and } t_2 \geq p + 1 \\ 0 & \text{otherwise.} \end{cases}$$

which implies that

$$\max_{\substack{t_1 < t_2 \\ \Delta^q(i, t_1, t_2) > 0}} \left(t_2 + \left\lceil \frac{1}{c_i} \Delta^q(i, t_1, t_2) \right\rceil - p_i \right) \quad = \quad q + 1.$$

The length of the sequence returned $\texttt{Propagate}(\mathcal{P}, (W\text{-R})^\star)$ is thus at least $p+1$. As a consequence, we have exhibited a family of DRPs with 2 tasks where the number of applications of the pruning operators is not bounded by any function of n. ■

Corollary 4 *If W is a reasonable energy function, then the pattern $(W\text{-}D)^\star$ is not horizon-independent for DRPs.*

6 Cumulative Scheduling

We now consider the cumulative case, which is known to be harder than the disjunctive in practice. Obviously, by Theorem 6, the energetic fixpoints are pseudo-polynomial. It remains to consider the edge-finding fixpoints. Our first result is positive and shows that (EF-R)* and (EF-D)* are strongly polynomial.

Theorem 7 *The complexity of the propagation patterns (EF-R)* and (EF-D)* is $O(n)$.*

Proof (Sketch) The full proof for (EF-R)* is in Appendix C. The proof shows that, at iteration q of the fixpoint, there are at least $q+1$ tasks whose release date have reached their final value. As a consequence, the fixpoint must be reached at iteration $n-1$.

The first step in the proof identifies why a release date can be updated at iteration $q+1$. As in the disjunctive case, there are two (non-exclusive) reasons:

1. either the release date of a task j with an earlier due date has been modified at iteration q, which may induce an update to task i when the condition $\alpha(\Omega, i)$ holds for a set Ω including j;

2. or the release of tasks i was already updated at iteration q but that update used a set Ω_q whose due date is smaller than the due date of the set Ω_{q+1} used at iteration $q+1$.

As due dates are not modified during this process, it is natural to partition T in subsets of tasks with the same due dates and to order these subsets $(T_k)_{k=1,\ldots,K}$ by increasing due dates. The second step then proves by induction that, at iteration k, all tasks of $T_1 \cup \ldots \cup T_{k+1}$ have reached their final state. ■

Unfortunately, the propagation pattern (EF-R $\cdot$ EF-D)* is not polynomial and suffers from a ping-pong effect between the release and the due dates.

Theorem 8 *The pattern (EF-R $\cdot$ EF-D)* is not horizon-independent for CRPs.*

Proof Let $p \geq 3$. Consider the following CRP

task	r	d	p	c
$toLeft$	-2p	p	2p + 1	1
$toRight$	-p	2p	2p + 1	1
$middle$	-2	+2	1	1
$left_A$	-2p	0	p - 1	1
$left_B$	-2p	0	p - 1	1
$right_A$	0	2p	p - 1	1
$right_B$	0	2p	p - 1	1
$igniter$	-2p	0	1	1

where the resource has capacity 2. For $0 \leq q \leq p-2$, all release dates and due dates of $\mathcal{P}_{2q}$ are those of $\mathcal{P}_0$ except

$$r^{2q}_{toRight} = q - p \qquad\qquad d^{2q}_{toLeft} = p - q.$$

As a consequence, the length of `Propagate`($\mathcal{P}$, (EF-R $\cdot$ EF-D)*) is greater or equal to $2(p-2)$ so the pattern is not polynomial. ■

Here is an intuitive explanation of this propagation. Without task *igniter*, the instance is invariant under time symmetry. The addition of *igniter* breaks this balance and induces the edge finder to modify the release date of *toRight*. This leads to a series of updates without 'energy loss' because of the symmetry between $toLeft$ and *toRight*, thus making this propagation self-sustained.

Note that this is a very robust counter-example. For example, reasonable energetic operators behave as the edge-finder on this instance. Moreover, we have tried several variants of these operators, not defined in this paper, which also suffered from this ping-pong effect. Thus, it is reasonable to consider this instance as a good benchmark for future pruning operators. A fast convergence on this instance would provide hope that the considered pattern is polynomial.

Corollary 5 *No fixpoint pattern over {EF-R, EF-D} is horizon-independent for CRPs.*

Proof Direct consequence of Theorems 8 and 2. ■

7 Conclusion and Open Questions

The complexity analysis of edge finding and energetic operators has focused on a single application of these operators. Such analyses are problematic, since these operators typically interact in the fixpoint computation of constraint-based schedulers and are not idempotent so that a single application does not make much sense. As a consequence, available analyses leave a significant gap in our understanding of these pruning operators and do not provide a sound basis for algorithmic comparison.

This paper addressed this limitation and studied the complexity of propagation algorithms for disjunctive and cumulative resources. It contains three main results that may be summarized as follows:

- For disjunctive resources, the fixpoint of the edge finder for both release and due dates can be reached in strongly polynomial time.
- For the cumulative resources, the fixpoint can be reached in strongly polynomial time for updating the release dates or the due dates but not both.
- The fixpoint of "reasonable" energetic operators cannot be reached in strongly polynomial time, even for disjunctive scheduling and even when only the release dates or the due dates are considered.

In addition, the paper presented a generic propagation algorithm with two fundamental properties:

1. it is guaranteed to be strongly polynomial if such an algorithm exists;

2. an optimal constraint propagation algorithm can never be much more efficient than the generic algorithm.

This research also opens a variety of open issues.

- Is the complexity bound in Theorem 5 tight? We conjecture that it is not, because of the underlying structure of precedence constraints.
- Are the "not first/not last" algorithms for disjunctive scheduling strongly polynomial? These algorithms [BLPN01, TL00, Vil04] are often used with edge finders and exploit different properties.
- Can the limitation of energetic operator be remedied? Energetic operators are good at finding pairs (t_2, i) such that i must ends after date t_2, but they badly exploit this information. Using a variant of the rest function found in the edge finder would address this limitation, but the complexity of the resulting operators should be analyzed.
- What is the complexity of patterns such as $(W_{Sh}\text{-R} \cdot \text{EF-R})^\star$ combining energetic reasoning and edge finding?
- More generally, does there exist stronger pruning operators that do not suffer from the inherent computational limitations of edge finding and energetic pruning operators in the cumulative case?

References

[AB93] A. Aggoun and N. Beldiceanu. Extending CHIP To Solve Complex Scheduling and Packing Problems. *Journal of Mathematical and Computer Modelling*, 17(7):57–73, 1993.

[BLPN01] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling*. Kluwer Academic Publishers, 2001.

[Bru95] *Scheduling algorithms*. Springer, Berlin, 1995.

[CL94] Y. Caseau and F. Laburthe. Improving CLP Scheduling with Task Intervals. In *Proceedings of the 11th International Conference on Logic Programming (ICLP'94)*, pages 369–383, Santa Margherita Ligure, Italy, 1994.

[CP94] J. Carlier and E. Pinson. Adjustment of Heads and Tails for the Jobshop Problem. *European Journal of Operational Research*, 78:146–161, 1994.

[LP94] C. Le Pape. Implementation of Resource Constraints in ILOG SCHEDULE: A Library for the Development of Constraint-Based Scheduling Systems. *Intelligent Systems Engineering*, 3(2):55–66, 1994.

[MS96] D. Martin and P. Shmoys. A Time-based Approach to the Job-Shop Problem. In *Proc. of 5th International Conference on Integer Programming and Combinatorial Optimization (IPCO'96)*, Vancouver, Canada, 1996. Springer Verlag.

[MVH05] L. Mercier and P. Van Hentenryck. Edge-finding for cumulative scheduling. *Submitted to publication*, 2005.

[NA96] W. Nuijten and E. Aarts. A Computational Study of Constraint Satisfaction for Multiple capacitated Job-Shop Scheduling. *European Journal of Operations Research*, 90:269–284, 1996.

[TL00] P. Torres and P. Lopez. On not-first/not-last conditions in disjunctive scheduling. *European Journal of Operational Research*, 127(2):332–343, 2000.

[VBv04] P. Vilim, R. Barták, and O. Čepek. Unary Resource Constraint with Optional Activities. In *Principles and Practice of Constraint Programming - CP 2004: 10th International Conference*, 2004.

[Vil04] Petr Vilim. O(n log n) Filtering Algorithms for Unary Resource Constraint. In *Proceedings of the First International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'04)*, pages 319–334, Nice, 2004.

Appendices

A Proof of Theorem 1

Theorem 1 *Let F_1 and F_2 be two fixpoint patterns over $\Psi = \{\psi_1, \ldots, \psi_k\}$ and $\mathcal{P}$ be a CRP. Then,* $\textsc{last}(\texttt{Propagate}(\mathcal{P}, F_1)) = \textsc{last}(\texttt{Propagate}(\mathcal{P}, F_2))$*, where* $\textsc{last}(S)$ *denotes the last element of a non-empty sequence.*

Proof Let

$$\langle \mathcal{P}_1^1, \ldots, \mathcal{P}_{Q_1}^1 \rangle = \texttt{Propagate}(\mathcal{P}, (F_1)^\star)$$

and

$$\langle \mathcal{P}_1^2, \ldots, \mathcal{P}_{Q_2}^2 \rangle = \texttt{Propagate}(\mathcal{P}, (F_2)^\star)$$

and define $\mathcal{P}_0^1 = \mathcal{P}_0^2 = \mathcal{P}$. By definition of a fixpoint, we have $\psi(\mathcal{P}_{Q_1}^1) = \mathcal{P}_{Q_1}^1$ and $\psi(\mathcal{P}_{Q_2}^2) = \mathcal{P}_{Q_2}^2$ for each pruning operator ψ in Ψ. By contractance of the operators, we also have

$$\mathcal{P}_0^1 \supseteq \cdots \supseteq \mathcal{P}_{Q_1}^1$$

and

$$\mathcal{P}_0^2 \supseteq \cdots \supseteq \mathcal{P}_{Q_2}^2.$$

Assume now that $\mathcal{P}_{Q_1}^1 \subseteq \mathcal{P}_{Q_2}^2$ does not hold and define

$$S = \left\{ 0 \leq q \leq Q_2 \;\middle|\; \mathcal{P}_{Q_1}^1 \subseteq \mathcal{P}_q^2 \right\}.$$

S is not empty since $\mathcal{P}_{Q_1}^1 \subseteq \mathcal{P}_0^1 = \mathcal{P}_0^2$. Now define $\overline{q} = \max S$. Since $\mathcal{P}_{Q_1}^1 \subseteq \mathcal{P}_{Q_2}^2$ does not hold, it follows that $\overline{q} < Q_2$. As a consequence, there exists $\psi \in \Psi$ such that $\mathcal{P}_{\overline{q}+1}^2 = \psi\left(\mathcal{P}_{\overline{q}}^2\right)$. By monotonicity of ψ, it follows that $\psi\left(\mathcal{P}_{Q_1}^1\right) \subseteq \psi\left(\mathcal{P}_{\overline{q}}^2\right) = \mathcal{P}_{\overline{q}+1}^2$. Since $\psi(\mathcal{P}_{Q_1}^1) = \mathcal{P}_{Q_1}^1$, $\overline{q}+1 \in S$, which contradicts the hypothesis. Hence $\mathcal{P}_{Q_1}^1 \subseteq \mathcal{P}_{Q_2}^2$. The result follows from the other inclusion whose proof is similar. ■

B Proof of Theorem 4

Theorem 4 *Let F be a propagation pattern. There exist 2 constants a and b satisfying $0 < a \leq 1$ and $b \geq 1$ such that, for any CRP $\mathcal{P}_0$, we have*

$$\left|\left\{ 0 \leq q < Q \;\middle|\; \mathcal{P}_{q+1} \neq \mathcal{P}_q \right\}\right| \quad \geq \quad aQ - b$$

where

$$\langle \mathcal{P}_1, \ldots, P_Q \rangle = \texttt{Propagate}(\mathcal{P}_0, F).$$

Proof The proof is by induction on the structure of the propagation pattern. The theorem holds for patterns of the form ψ with $a = 1$ and $b = 1$.

Consider now the sequential composition $F_1 \cdot F_2$ and assume that F_1 and F_2 verify the theorem with (a_1, b_1) and (a_2, b_2) respectively. The sequence

$$\langle \mathcal{P}_1, \ldots, \mathcal{P}_{Q_1}, \mathcal{P}_{Q_1+1}, \ldots, \mathcal{P}_{Q_2} \rangle$$

returned by $\texttt{Propagate}(\mathcal{P}_0, F_1 \cdot F_2)$ is obtained by applying $\texttt{Propagate}(\mathcal{P}_0, F_1)$ giving

$$\langle \mathcal{P}_1, \ldots, \mathcal{P}_{Q_1} \rangle$$

and by applying $\texttt{Propagate}(\mathcal{P}_{Q_1}, F_2)$ to obtain

$$\langle \mathcal{P}_{Q_1+1}, \ldots, \mathcal{P}_{Q_2} \rangle.$$

By induction hypothesis, we have

$$\begin{aligned} |\{0 \leq q < Q_1 \mid \mathcal{P}_{q+1} \neq \mathcal{P}_q\}| &\geq a_1 Q_1 - b_1 \\ |\{Q_1 \leq q < Q_2 \mid \mathcal{P}_{q+1} \neq \mathcal{P}_q\}| &\geq a_2(Q_2 - Q_1) - b_2 \end{aligned}$$

It follows that

$$|\{0 \leq q < Q_2 \mid \mathcal{P}_{q+1} \neq \mathcal{P}_q\}| \geq a_1 Q_1 + a_2(Q_2 - Q_1) - (b_1 + b_2)$$

and the theorem holds by choosing $a = \min(a_1, a_2)$ and $b = b_1 + b_2$.

Finally, consider the fixpoint operator $(F)^\star$ and assume that F verifies the theorem with (a, b). Let

$$\langle \mathcal{P}_1, \ldots, \mathcal{P}_{Q_1}, \ldots, \mathcal{P}_{Q_{k-1}}, \ldots, \mathcal{P}_{Q_k} \rangle$$

be the sequence returned by $\texttt{Propagate}(\mathcal{P}_0, (F)^\star)$ where

$$\langle \mathcal{P}_{Q_i+1}, \ldots, \mathcal{P}_{Q_{i+1}} \rangle$$

is the sequence returned by $\texttt{Propagate}(\mathcal{P}_{Q_i}, (F)^\star)$. Define

$$\mathcal{A}_i = \{Q_i \leq q < Q_{i+1} \mid \mathcal{P}_{q+1} \neq \mathcal{P}_q\}$$

and

$$\mathcal{A} = \bigcup_i \mathcal{A}_i.$$

Because Line 12 in Algorithm `Propagate` is only executed when the test in line 9 fails, it follows that

$$\forall i \leq k-2,\ |\mathcal{A}_i| \geq 1 \qquad \text{and} \qquad |\mathcal{A}_{k-1}| = 0.$$

We first prove that

$$\sum_{i \leq k-2} |\mathcal{A}_i| \geq \frac{a}{2b} Q_{k-1}.$$

Consider $i \leq k-2$ and assume that $Q_{i+1} - Q_i > \frac{2b}{a}$. Since

$$|\mathcal{A}_i| \geq a(Q_{i+1} - Q_i) - b,$$

we have

$$\frac{|\mathcal{A}_i|}{Q_{i+1} - Q_i} \;\geq\; a - \frac{b}{Q_{i+1} - Q_i} \;>\; a - \frac{a}{2} \;=\; \frac{a}{2}.$$

Consider $i \leq k-2$ and assume that $Q_{i+1} - Q_i \leq \frac{2b}{a}$. Since $|\mathcal{A}_i| \geq 1$, we have

$$\frac{|\mathcal{A}_i|}{Q_{i+1} - Q_i} \;\geq\; \frac{1}{Q_{i+1} - Q_i} \;\geq\; \frac{a}{2b}$$

Since $b \geq 1$, the two cases lead to

$$\sum_{i \leq k-2} |\mathcal{A}_i| \geq \frac{a}{2b} Q_{k-1}.$$

It remains to study the last application of F. Since $\mathcal{A}_{k-1} = 0$, it follows by induction that $0 \geq a(Q_k - Q_{k-1}) - b$. Thus $Q_k - Q_{k-1} \leq \frac{b}{a}$ and it follows that

$$|\mathcal{A}| \;\geq\; \frac{a}{2b}(Q_k - (Q_k - Q_{k-1})) \;\geq\; \frac{a}{2b} Q_k - \frac{a}{2b} \cdot \frac{b}{a} \;=\; \frac{a}{2b} Q_k - \frac{1}{2}.$$

As a consequence, the pattern $(F)^\star$ verifies the theorem with $a^\star = \frac{a}{2b}, b^\star = 1$. ■

C Proof of Theorem 7

The proof of Theorem 7 requires several lemmas and some additional notations. Given a CRP $\mathcal{P} = (r, d)$, our goal is to study the sequence $(r^q)_{q \in \mathbb{N}}$ defined by

$$(\text{EF-R})^q(\mathcal{P}) = (r^q, d)\,.$$

We use the following notations with capture the relevant conditions and formulas for position q in the sequence.

$$\begin{aligned}
r^q_\Omega \quad &= \quad \min\left\{ r^q_j \mid j \in \Omega \right\}, \\
\alpha^q(\Omega, i) \quad &\iff \quad \left(C(d_\Omega - r^q_{\Omega \cup \{i\}}) < e_{\Omega \cup \{i\}} \right), \\
\text{rest}^q(\Omega, c) \quad &= \quad \begin{cases} 0 & \text{if } \Omega = \emptyset \\ e_\Omega - (C - c)(d_\Omega - r^q_\Omega) & \text{otherwise.} \end{cases}
\end{aligned}$$

The next definition captures the concept of maximal valid pair for a task i, i.e., the pair (Ω, Θ) used to update the release date of task i at some iteration (if any).

Definition 15 (Valid Pair) *Let $i \in T$. A pair (Ω, Θ) is i-valid at iteration q if*

$$\left[i \notin \Omega\right] \ \wedge \ \left[\alpha^q(\Omega, i)\right] \ \wedge \ \left[\Theta \subseteq \Omega\right] \ \wedge \ \left[\mathrm{rest}^q(\Theta, c_i) > 0\right].$$

A i-valid pair (Ω, Θ) at iteration q is maximal if it satisfies

$$r_i^{q+1} = r_\Theta^q + \left\lceil \frac{1}{c_i} \mathrm{rest}^q(\Theta, c_i) \right\rceil.$$

The next lemma captures an important property of the edge finder. It specializes Proposition 7 in [MVH05] whose proof is long and technical, and not reproduced here.

Lemma 1 *Let $\mathcal{P} = (r, d)$ be a CRP, and (r', d) be (EF-R)($\mathcal{P}$). Then, for all $i \in T$, for all Ω such that $\alpha(\Omega, i)$, for all $\Theta \subseteq T$ such that $d_\Theta \leq d_\Omega$*

$$\mathrm{rest}(\Theta, c_i) > 0 \implies r'_i \geq r_\Theta + \left\lceil \frac{1}{c_i} \mathrm{rest}(\Theta, c_i) \right\rceil.$$

The next lemma specifies the two conditions under which a release date can be updated. These conditions were mentioned in the proof sketch but are formalized here.

Lemma 2 *Let $i \in T$ and $q \in \mathbb{N}$ be such that $r_i^{q+2} > r_i^{q+1}$. Suppose that $\mathcal{P}_{q+2} \neq \mathcal{P}_\emptyset$. Then at least one of the following statements is true:*

1. *For all maximal i-valid pair (Ω^+, Θ^+) at iteration $q+1$, there exists a task $j \in \Omega^+$ such that $r_j^{q+1} > r_j^q$;*
2. *$r_i^{q+1} > r_i^q$ and there exists a maximal i-valid pair (Ω, Θ) at iteration q such that for all maximal i-valid pair (Ω^+, Θ^+) at iteration $q+1$, $d_{\Omega^+} > d_\Omega$.*

Proof As $r_i^{q+2} > r_i^{q+1}$, there exists a maximal i-valid pair at iteration $q+1$. Each such pair (Ω^+, Θ^+) verifies

$$\begin{cases} \alpha^{q+1}(\Omega^+, i) \\ d_{\Theta^+} \leq d_{\Omega^+} \\ \mathrm{rest}^{q+1}(\Theta^+, c_i) > 0 \\ r_i^{q+2} = r_{\Theta^+}^{q+1} + \left\lceil \frac{1}{c_i} \mathrm{rest}^{q+1}(\Theta^+, c_i) \right\rceil \end{cases}$$

First observe that $d_i > d_{\Omega^+}$. Otherwise, since $\alpha^{q+1}(\Omega^+, i)$ holds,

$$e_{\Omega^+ \cup \{i\}} > C\left(d_{\Omega^+ \cup \{i\}} - r_{\Omega^+ \cup \{i\}}\right)$$

and $\mathcal{P}_{q+2} = \mathcal{P}_\emptyset$ which contradicts the lemma hypothesis.

Assume now that Statement (1) does not hold. That implies the existence of a maximal i-valid pair (Ω^+, Θ^+) at $q+1$ satisfying

$$\forall j \in \Omega^+ : r_j^{q+1} = r_j^q.$$

Since $\Theta^+ \subseteq \Omega^+$, we have

$$\text{rest}^q(\Theta^+, c_i) > 0 \tag{1}$$

$$r^q_{\Theta^+} + \left\lceil \frac{1}{c_i} \text{rest}^q(\Theta^+, c_i) \right\rceil = r^{q+1}_{\Theta^+} + \left\lceil \frac{1}{c_i} \text{rest}^{q+1}(\Theta^+, c_i) \right\rceil \tag{2}$$

As a consequence, by definition of the edge-finding operator, the fact that $r_i^{q+1} < r_i^{q+2}$ means that the precondition $\alpha^q(\Omega^+, i)$ did not hold at iteration q. Thus

$$\begin{array}{rcll}
C\left(d_\Omega - r^{q+1}_{\Omega^+\cup\{i\}}\right) & < & e_{\Omega^+\cup\{i\}} & \text{because } \alpha^{q+1}(\Omega^+, i) \\
C\left(d_\Omega - r^{q}_{\Omega^+\cup\{i\}}\right) & \geq & e_{\Omega^+\cup\{i\}} & \text{because } \neg\alpha^{q}(\Omega^+, i) \\
r^{q+1}_{\Omega^+\cup\{i\}} & > & r^{q}_{\Omega^+\cup\{i\}} & \text{by transitivity} \\
\min\left(r_i^{q+1},\ r^{q+1}_{\Omega^+}\right) & > & \min\left(r_i^{q},\ r^{q}_{\Omega^+}\right) & \text{by definition.}
\end{array}$$

But, since $r^{q+1}_{\Omega^+} = r^q_{\Omega^+}$, the minimum of the right-hand side is reached by r_i^q and we have that

$$r_i^{q+1} > r_i^q.$$

It remains to show the second part of statement 2 in the lemma. Consider a maximal i-valid pair (Ω, Θ) at iteration q and assume that $d_\Omega \geq d_{\Omega^+}$. Since (Ω^+, Θ^+) was valid at iteration $q+1$, $d_{\Theta^+} \leq d_{\Omega^+}$ and thus $d_{\Theta^+} \leq d_\Omega$. Since $\alpha^q(\Omega, i)$ holds, it follows from Lemma 1 that

$$r_i^{q+1} \geq r^q_{\Theta^+} + \left\lceil \frac{1}{c_i} \text{rest}^q(\Theta^+, c_i) \right\rceil$$

By inequality (1) and equation (2), this implies $r_i^{q+1} = r_i^{q+2}$, which contradicts the hypothesis. ■

Lemma 2 demonstrates that the due dates impose a natural ordering of the tasks (Statement 2). This observation is the foundation of the remaining part of the proof. The key idea is to partition the tasks by due dates

$$D_1 < \ldots < D_K$$

and to show that the tasks in T_k with due date D_k reach their final release dates at iteration $k-1$. More precisely, we define $\mathcal{D} = \{d_i \mid i \in T\}$, $K = |\mathcal{D}|$, and $\mathcal{D} = \{D_1, \ldots, D_K\}$ with$D_1 < \cdots < D_K$. We also partition T into

$$T = \biguplus_{1 \leq k \leq K} T_k$$

where

$$T_k = \{ i \in T \mid d_i = D_k \}.$$

Finally, we define

$$\overline{T}_k = \biguplus_{1 \leq l \leq k} T_l.$$

Our next lemma bounds the iteration when a release date reaches its final value. Intuitively, in the proof, $\varphi(k)$ denotes the earliest iteration at which a task with due date D_k or earlier reaches its final release date.

Lemma 3 *With the above notations, let $\mathcal{P}$ be a CRP such that $\mathcal{P}_K \neq \mathcal{P}_\emptyset$ and define*

$$\varphi(k) = \min \left\{ q \in \mathbb{N} \;\middle|\; \forall r \in \mathbb{N},\ \forall i \in \overline{T}_k,\ r_i^{q+r} = r_i^q \right\}.$$

Then, for all $0 \leq k \leq K-1$, we have $\varphi(k+1) \leq k$.

Proof The proof is by induction with a stronger induction hypothesis. We show that, for all $0 \leq k \leq K-1$, we have

(A_k) $\varphi(k+1) \leq k$;

(B_k) For $i \in T$ such that $r_i^{k+1} > r_i^k$, for each maximal i-valid pair (Ω, Θ) at k, $d_\Omega \geq D_{k+1}$.

For the base case $k = 0$, we have $\varphi(1) = 0$, because the release dates of tasks of T_1 cannot be improved without discovering a contradiction. Moreover, for any valid pair (Ω, Θ), Ω is not empty and thus $d_\Omega \geq D_1$ since $D_1 = \min \{ d_i \mid i \in T \}$.

Consider now the inductive case and assume that (A_k) and (B_k) hold for a given k. We first show that (B_{k+1}) holds. Consider $i \in T$ such that $r_i^{k+2} > r_i^{k+1}$ and suppose there exists a maximal i-valid pair (Ω^+, Θ^+) at $k+1$ such that $d_{\Omega^+} \leq D_{k+1}$. By Lemma 2, two cases must be considered.

1. Assume first that Statement (1) of Lemma 2 holds. Then, there exists $j \in \Omega^+$ such that $r_j^{k+1} > r_j^k$. Since $j \in \Omega^+$, $d_j \leq d_{\Omega^+} \leq D_{k+1}$ and hence $j \in \overline{T}_{k+1}$ which contradicts (A_k).

2. Assume now that statement (2) of Lemma 2 holds. Let (Ω^+, Θ^+) and (Ω, Θ) be the maximal i-valid pairs chosen at iterations $k+1$ and k, according to statement (2). By (B_k), $d_\Omega \geq D_{k+1}$ and, by Lemma 2, $d_{\Omega^+} > d_\Omega$. It follows that $d_{\Omega^+} > D_{k+1}$, which contradicts the hypothesis.

This ends the proof of (B_{k+1}).

It remains to prove (A_{k+1}). Let $i \in \overline{T}_{k+2}$. Then, as no contradiction is discovered, any update of the release date of i can only be made by valid pairs (Ω, Θ) such that

$d_\Omega < d_i$. Hence $d_\Omega \leq D_{k+1}$. As a consequence, by Property (B_{k+1}), such an update cannot occur at iteration $k+1$ and

$$\forall i \in \overline{T}_{k+2} : \; r_i^{k+2} = r_i^{k+1}.$$

As tasks of $T \setminus \overline{T}_{k+2}$ have no influence on the updates of release dates of tasks of $\overline{T}_{k+2}$, the release dates of the tasks in $\overline{T}_{k+2}$ have reached their final values at iteration $k+1$. It follows that $\varphi(k+2) \leq k+1$, proving (A_{k+1}). ■

Theorem 7 *The complexity of the propagation patterns (EF-R)* and (EF-D)* is $O(n)$.*

Proof Let $\phi(\mathcal{P})$ be defined as folows:

$$\phi(\mathcal{P}) = \min \left\{ q \in \mathbb{N} \;\middle|\; r^{q+1} = r^q \right\}.$$

By Lemma 3, $\phi(\mathcal{P}) = \varphi(K)$. Since $K \leq n$, all release dates have reached their final values at iteration n. Since the sequence returned by `Propagate` $(\mathcal{P}, (\text{EF-R})^\star)$ has length $\phi(\mathcal{P}) + 1$, the complexity of $(\text{EF-R})^\star$ is $O(n)$. A similar result holds for $(\text{EF-D})^\star$ by Corollary 1. ■

D Proof of Theorem 5

A key difference with the disjunctive and cumulative edge finders is that the fact that, in the disjunctive case, the function EF-R (resp. EF-D) only depends on the due dates (resp. release dates) in the condition α and not the actual expression of the lower bound. This difference prevents the ping-pong effect exhibited in Theorem 8 from occurring in disjunctive scheduling, since it relies on iterative updates for a specific pair (Ω, i). The proof described below formalizes this insight.

Our first lemma in this section provides an alternative, but equivalent, definition of the edge finder in the disjunctive case. The new definition is motivated by the difficulty of manipulating condition α in the edge finder. Indeed, it is not necessarily the case that $\alpha(\Omega_1 \cup \Omega_2, i)$ holds when $\alpha(\Omega_1, i)$ and $\alpha(\Omega_2, i)$ hold, which complicates the proof. The new formulation removes this issue.

Lemma 4 *Let $\mathcal{P} = (r, d)$ be a DRP. Let $\Gamma^- : T \to 2^T$ be defined by*

$$\Gamma^-(i) = \bigcup_{\substack{i \notin \Omega \subseteq T \\ \alpha(\Omega, i)}} \Omega.$$

Let $LBU : T \to \mathbb{Z}$ be the function defined by

$$LBU_i = \max_{\Theta \subseteq \Gamma^-(i)} r_\Theta + p_\Theta.$$

We have that $LBU = LB$.

Proof The inequality $LB \leq LBU$ is immediate and we show that $LBU \leq LB$. Consider a task $i \in T$. If $LBU_i = -\infty$, then $\Gamma^-(i) = \emptyset$ and $LB_i = -\infty$. Assume now $LBU_i > -\infty$, in which case there exists $\Theta \subseteq \Gamma^-(i)$ such that $LBU_i = r_\Theta + p_\Theta$.

Let $j \in \Theta$ be a task such that $d_\Theta = d_j$. Since $j \in \Gamma^-(i)$, there exists $\Omega \subseteq T$ which contains j and satisfies $\alpha(\Omega, i)$. Since all tasks of Ω are in $\Gamma^-(i)$, we have that $LBU_i \geq r_\Omega + p_\Omega$. Moreover, $d_\Theta \leq d_\Omega$, since $j \in \Omega$ and $d_j = d_\Omega$. Now consider two cases:

case $\mathbf{r_\Theta \geq r_{\Omega \cup \{i\}}}$. Let $\Omega' = \Theta \cup \Omega$. We have

$$\begin{array}{rcll} r_{\Omega' \cup \{i\}} & = & r_{\Omega \cup \{i\}} & \text{because } r_\Theta \geq r_{\Omega \cup \{i\}} \\ d_{\Omega'} & = & d_\Omega & \text{because } d_\Theta \leq d_\Omega \\ p_{\Omega'} & \geq & p_\Omega & \text{because } \Omega \subseteq \Omega' \end{array}$$

Hence, since $\alpha(\Omega, i)$ holds, $r_{\Omega \cup \{i\}} + p_{\Omega \cup \{i\}} > d_\Omega$ which implies $r_{\Omega' \cup \{i\}} + p_{\Omega' \cup \{i\}} > d_{\Omega'}$ and thus $\alpha(\Omega', i)$ holds.

case $\mathbf{r_\Theta < r_{\Omega \cup \{i\}}}$. In that case, $r_{\Theta \cup \{i\}} = r_\Theta$. Thus

$$\begin{array}{rcll} r_{\Theta \cup \{i\}} + p_\Theta & = & r_\Theta + p_\Theta & \text{because } r_{\Theta \cup \{i\}} = r_\Theta \\ & \geq & r_\Omega + p_\Omega & \text{because } LBU_i \geq r_\Omega + p_\Omega \\ & > & d_\Omega - p_i & \text{because } \alpha(\Omega, i) \text{ and } r_\Omega \geq r_{\Omega \cup \{i\}} \\ & > & d_\Theta - p_i & \text{because } d_\Theta \leq d_\Omega \end{array}$$

Thus $\alpha(\Theta, i)$ holds.

In both cases, Θ is included in a set (either Ω' or itself) which satisfies property $\alpha(., i)$. Thus $LB_i \geq LBU_i$ and the result follows. ■

The rest of the proof analyzes the sequence of DRPs

$$(\mathcal{P}_q)_{q=0,\ldots,Q}.$$

It considers the set of iterations $A \subseteq \{0, \ldots, Q-1\}$ satisfying

$$\forall q \in A : \ \mathcal{P}_{q+1} = (\text{EF-R})(\mathcal{P}_q)$$

and such that $\forall q \notin A$, there exists an operator ψ which does not modify release dates and such that $\mathcal{P}_{q+1} = \psi(\mathcal{P}_q)$. We also define the sequence $\left(\Gamma_q^-\right)_{q=0,\ldots,Q}$ as

$$\Gamma_q^-(i) = \bigcup_{\substack{\Omega \subseteq T \\ \alpha^q(\Omega, i) \\ i \notin \Omega}} \Omega.$$

Lemma 5 *With the notations above,* $\left(\Gamma_q^-\right)$ *is non-decreasing.*

Proof Recall that $\alpha^q(\Omega, i) \iff d^q_\Omega - r^q_{\Omega\cup\{i\}} < p_{\Omega\cup\{i\}}$. Since processing times are constant, release dates are non-decreasing, and due dates are non-increasing, the fact that $\alpha^q(\Omega, i)$ holds implies that, for all $r > q$, $\alpha^r(\Omega, i)$ holds too. ■

Our final lemma bounds the number of iterations which update release dates without discovering new conditions of the form $\alpha(\Omega, i)$.

Lemma 6 *With the notations above, let* $q_1 < q_2$ *be such that* $\Gamma^-_{q_1} = \Gamma^-_{q_2}$. *Define*

$$Q_A = \left\{ q \in \{q_1, \ldots, q_2 - 1\} \ \middle| \ (q \in A) \wedge (\mathcal{P}_{q+1} \neq \mathcal{P}_q) \right\}$$

We have $|Q_A| \quad \leq \quad n - 1$.

Proof Consider the sequence $\langle \mathcal{P}'_0, \ldots, \mathcal{P}'_{|Q_A|} \rangle$ defined as

$$\begin{array}{lcl} \mathcal{P}'_0 & = & \mathcal{P}_{q_1} \\ \mathcal{P}'_{q+1} & = & (\text{EF-R})(\mathcal{P}'_q) \end{array}$$

where $\mathcal{P}'_i = (r'^i, d'^i)$. In other words,

$$\left(\mathcal{P}'_i\right)_{i \in 0..|Q_A|}$$

is the sequence obtained by keeping only operator EF-R and removing all other operators that do not affect release dates. Define the function f to establish the correspondence between the DRPs $\mathcal{P}_q$ and $\mathcal{P}'_{f(q)}$ as follows:

$$f(q) = |A \cap \{q_1, \ldots, q-1\}| \, .$$

We first prove by induction that

$$\forall q_1 \leq q \leq q_2 : \ r^q = r'^{f(q)}.$$

Intuitively, this means that only operator EF-R has an impact on the release dates and the other operators can be ignored. This holds for $q = q_1$ by definition of $(\mathcal{P}'_q)$. Suppose that it holds for a given q $(q_1 \leq q < q_2)$. If $q \notin A$, then $r^{q+1} = r^q$ and $f(q+1) = f(q)$, since the pruning operator applied at q does not change release dates. And . Hence $r^{q+1} = r'^{f(q+1)}$. Suppose now that $q \in A$. Since $r^q = r'^{f(q)}$ and $d^q \leq d'^{f(q)}$, $\alpha'^{f(q)}(\Omega, i) \Rightarrow \alpha^q(\Omega, i)$ for every pair (Ω, i). Hence, $\Gamma'^-_{f(q)} \subseteq \Gamma^-_q = \Gamma^-_{q_1}$. Since Γ'^- is non-decreasing, $\Gamma'^-_{f(q)} = \Gamma'^-_0$. By applying Lemma 4, once on the sequence $(\mathcal{P})$, and once on the sequence $(\mathcal{P}')$, we obtain

$$r_i^{q+1} = \max_{\Theta \subseteq \Gamma^-_q(i)} \left(r^q_\Theta + p_\Theta\right) = \max_{\Theta \subseteq \Gamma^-_{q_1}(i)} \left(r^q_\Theta + p_\Theta\right)$$

$$r_i'^{f(q+1)} = \max_{\Theta \subseteq \Gamma'^-_{f(q)}(i)} \left(r'^{f(q)}_\Theta + p_\Theta\right) = \max_{\Theta \subseteq \Gamma'^-_0(i)} \left(r'^0_\Theta + p_\Theta\right)$$

and thus $r_i^{q+1} = r_i'^{f(q+1)}$. This ends the proof of the relationships between $(\mathcal{P})$ and $(\mathcal{P}')$.

Finally, by Lemma 3, the sequence $(\mathcal{P}')$ reaches its fixpoint in at most $n-1$ iterations. Translated back to the sequence $(\mathcal{P})$, this implies that

$$\left|\{q \in \{q_1, \ldots, q_2 - 1\} \mid (q \in A) \wedge (\mathcal{P}_{q+1} \neq \mathcal{P}_q)\}\right| \leq n - 1. \qquad \blacksquare$$

Theorem 5 *Let Ψ be a set of pruning operators containing EF-R such that any other operator in Ψ do not modify release dates, i.e.,*

$$\forall\ \psi \in \Psi \setminus \{\textit{EF-R}\}, \forall\ \mathcal{P} = (r, d) :\ \exists\ d' \in \mathbb{Z}^n :\ \psi(\mathcal{P}) = (r, d').$$

Let F be a propagation pattern on Ψ, $\mathcal{P}_0$ be a DRP, and

$$\langle \mathcal{P}_1, \ldots, \mathcal{P}_Q \rangle = \texttt{Propagate}(\mathcal{P}_0, F).$$

Denote by

$$\langle \psi_0, \ldots, \psi_{Q-1} \rangle$$

the sequence of pruning operators applied to produce $\langle \mathcal{P}_1, \ldots, \mathcal{P}_Q \rangle$, i.e.,

$$\psi_q(\mathcal{P}_q) = \mathcal{P}_{q+1}.$$

We have that

$$\left|\{0 \leq q < Q \mid \psi_q = \textit{EF-R} \wedge \mathcal{P}_{q+1} \neq \mathcal{P}_q\}\right| \leq n^3.$$

Proof Let $(Q_h)_{h \geq 0}$ be the increasing sequences of integers such that $\Gamma^-_{Q_h} \neq \Gamma^-_{Q_h - 1}$. Since Γ^- is non decreasing (as stated by Lemma 5) and since, for each task i, no

more than $n-1$ tasks can belong to $\Gamma^-(i)$, the number of such integers (Q_h) is less or equal to n^2. Moreover, on the interval $Q_h, \ldots, Q_{h+1}-1$, Lemma 6 shows that there are no more than $n-1$ iterations where EF-R is applied and modify at least one release date. It follows that

$$\left|\{0 \leq q < Q \mid \psi_q = \text{EF-R} \wedge \mathcal{P}_{q+1} \neq \mathcal{P}_q\}\right| \quad \leq \quad n^3. \qquad \blacksquare$$

Appendix

The following paper is given in appendix
as it is cited in the report and is not yet published.
It is independent from the report, but reading it
can provide a better understanding of the edge-finder.

Edge Finding for Cumulative Scheduling

Luc Mercier and Pascal Van Hentenryck
Brown University, Box 1910, Providence, RI 02912
Email: {mercier,pvh}@cs.brown.edu

May 27, 2005

Abstract

Edge-finding algorithms for cumulative scheduling are at the core of commercial constraint-based schedulers. This paper shows that Nuijten's edge finder for cumulative scheduling, and its derivatives, are incomplete and use an invalid dominance rule. The paper then presents a new edge-finding algorithm for cumulative resources which runs in time $O(n^2k)$, where n is the number of tasks and k the number of different capacity requirements of the tasks. The new algorithm is organized in two phases and first uses dynamic programming to precompute the innermost maximization in the edge-finder specification. Finally, this paper also proposes the first extended edge-finding algorithm that runs in time $O(n^2k)$, improving the running time of available algorithms.

1 Introduction

Edge finding [CP94] is a fundamental pruning technique for disjunctive and cumulative scheduling[1] and is an integral part of commercial constraint-based schedulers. Informally speaking, an edge finder considers one resource at a time, identifies pairs (Ω, i) such that task i cannot precede (resp. follow) any task from Ω in all feasible schedules, and updates the earliest starting date (resp. latest finishing date) of task i accordingly. An edge-finding algorithm is a procedure that performs all such deductions.

Edge finding is well-understood for unary resources, i.e., resources with capacity one. Indeed, there exist efficient algorithms running in time $O(n \log n)$ or $O(n^2)$, where n is the number of tasks on the resource [CP94, Nui94, Vil04]. Edge finding is more challenging for cumulative resources whose capacity is a natural number $C \geq 1$ and whose tasks may require several capacity units. Nuijten [Nui94] (see also [NA96, BLPN01]) proposed an edge-finding algorithm running in time $O(n^2k)$, where $k \leq n$ is the number of distinct capacity requirements of the tasks. This algorithm was later refined to run in $O(n^2)$ [BLPN01].

This paper shows that Nuijten's algorithm, and its refinement, are incomplete and do not perform all the edge-finding updates. The mistake comes from the use of an incorrect dominance rule which holds for unary resources but does not carry over to the cumulative case. The paper also presents a new, two-phase, edge finder for cumulative resources that runs in $O(n^2k)$. The first phase is a

[1] This paper considers only non-preemptive problems, where tasks cannot be interrupted.

dynamic programming algorithm that precomputes the potential edge-finding updates. The second phase uses the precomputation to apply the actual updates. Moreover, similar ideas can be used to derive an $O(n^2k)$ for the extended edge-finding rule, improving the running time of the best available algorithms. The contributions of this paper can thus be summarized as follows:

1. This paper shows that Nuijten's algorithm and its derivatives are incomplete with respect to the edge-finding rule;
2. This paper presents a complete edge-finding algorithm that runs in time $O(n^2k)$;
3. This paper presents a complete extended edge-finding algorithm running in time $O(n^2k)$, improving the complexity of the best-known algorithm.

The rest of this paper is organized as follows. Section 2 specifies the problem and the notations used in the paper. Section 3 proves that Nuijten's algorithm is incomplete. Sections 4, 5, and 6 are the core of the paper: they present the dominance properties used for cumulative edge finding and the edge-finding algorithm itself. Section 7 presents the extended edge-finding algorithm, and Section 8 concludes the paper.

2 Problem Definition and Notations

Definition 1 (Cumulative Resource Problems) *A cumulative resource problem (CRP) is specified by a cumulative resource of capacity C and a set of tasks T. Each task $t \in T$ is specified by its release date r_t, its deadline d_t, its processing time p_t, and its capacity requirement c_t, all of which being natural numbers. A solution to a CRP $\mathcal{P}$ is a schedule that assigns a starting date s_t to each task t so that*

$$\forall t \in T : r_t \leq s_t \leq s_t + p_t \leq d_t$$

and

$$\forall i : \sum_{\substack{t \in T \\ s_t \leq i < s_t + p_t}} c_t \leq C.$$

The set of solutions to a CRP $\mathcal{P}$ is denoted by $sol(\mathcal{P})$. Finally, Sc denotes the set $\{c_t \mid t \in T\}$ of all capacity requirements, n denotes $|T|$, $N = \{1, \ldots, n\}$, k denotes $|Sc|$, and $e_t = c_t p_t$ denotes the energy of a task t.

In the following, we abuse notations and assume an underlying CRP with its resource and tasks specified as in Definition 1. We also lift the concepts of release dates, due dates, and energies to sets of tasks, i.e.,

$$r_\Omega = \min_{j \in \Omega} r_j$$
$$d_\Omega = \max_{j \in \Omega} d_j$$
$$e_\Omega = \sum_{j \in \Omega} e_j$$

where Ω is a set of tasks. By convention, when Ω is the empty set, $r_\Omega = \infty$, $d_\Omega = -\infty$ and $e_\Omega = 0$.
The CRP is NP-complete and constraint-based schedulers typically use a relaxation of feasibility to prune the search space.

Definition 2 (E-Feasibility) *A CRP is E-feasible if*

$$\forall \Omega \subseteq T : C(d_\Omega - r_\Omega) \geq e_\Omega.$$

Obviously, feasibility of a CRP implies E-feasibility. A critical aspect of constraint-based schedulers is to reduce the possible starting and finishing dates that appear in solutions. The edge-finding rule is one of the fundamental techniques to reduce these dates in disjunctive and cumulative scheduling. This paper restricts attention to starting dates only (the handling of finishing dates is similar), in which case the key idea underlying the edge-finding rule can be summarized as follows. Consider a set of tasks Ω and a task $i \in T \setminus \Omega$. If the condition

$$C(d_\Omega - r_{\Omega \cup \{i\}}) < e_{\Omega \cup \{i\}}$$

holds, then there exists no schedule in which task i precedes any operation in Ω. As a consequence, in any feasible schedule, the starting date s_i must satisfy

$$s_i \geq r_\Theta + \left\lceil \frac{1}{c_i} \operatorname{rest}(\Theta, c_i) \right\rceil$$

for all $\Theta \subseteq \Omega$ satisfying

$$\operatorname{rest}(\Theta, c_i) > 0$$

where

$$\operatorname{rest}(\Theta, c_i) = \begin{cases} e_\Theta - (C - c_i)(d_\Theta - r_\Theta) & \text{if } \Theta \neq \emptyset; \\ 0 & \text{otherwise.} \end{cases}$$

Informally speaking, $\operatorname{rest}(\Theta, c_i)$ is the energy of e_Ω that cannot be accommodated by a cumulative resource of capacity $C - c_i$ in the interval $[r_\Theta, d_\Theta)$. The proofs of these results can be found in [BLPN01]. We are now ready to specify the edge-finding algorithm.

Specification 1 (Edge Finding) *The edge-finding algorithm receives as input an E-feasible CRP. It produces as output a vector*

$$\left\langle \overline{LB_2}(1), \ldots, \overline{LB_2}(n) \right\rangle$$

where

$$\overline{LB_2}(i) = \max(r_i, LB_2(i))$$

and

$$LB_2(i) = \max_{\substack{\Omega \subseteq T \\ i \notin \Omega \\ \alpha(\Omega, i)}} \; \max_{\substack{\Theta \subseteq \Omega \\ \operatorname{rest}(\Theta, c_i) > 0}} r_\Theta + \left\lceil \frac{1}{c_i} \operatorname{rest}(\Theta, c_i) \right\rceil$$

with

$$\alpha(\Omega, i) \iff \left(C(d_\Omega - r_{\Omega \cup \{i\}}) < e_{\Omega \cup \{i\}} \right).$$

3 Incompleteness of Nuijten's Algorithm

We now consider algorithm CalcLB (Figure 4.9 in [Nui94]; see also [BLPN01]), which is is reproduced in Algorithm 1 for simplicity.[2] Nuijten claims that CalcLB computes $\overline{LB_2}(i)$ for all $i \in T$, which is incorrect. Consider the following instance on a resource of capacity 4:

task	r	d	p	c
a	0	69	4	1
b	1	2	1	4
c	0	3	1	2
d	0	3	1	2
e	2	3	1	1

Consider the pair (Ω, Θ) where $\Omega = T \setminus \{a\}$ and $\Theta = \{b\}$. The condition $\alpha(\Omega, a)$ holds because $e_{\Omega\cup\{a\}} = 13$ and $C(d_\Omega - r_{\Omega\cup\{a\}}) = 4 \times 3 = 12$. Moreover, we have $\Theta \subseteq \Omega$ and $\text{rest}(\Theta, c_a) = 1$ which implies

$$LB_2(a) \geq r_\Theta + \frac{1}{c_a}\,\text{rest}(\Theta, c_a) = 2.$$

Algorithm CalcLB does not perform this deduction because it never considers the pair (Ω, Θ). Instead, CalcLB considers the pair (Ω, Ω). But since $\text{rest}(\Omega, c_a) = 0$, no update takes place. The problem with CalcLB is apparent in line 7 which maintains l as the maximum due date of Ω. This maximal value is then used to compute (incorrectly) the rest in line 9, performing no update on the relevant g_j and thus no update on the release date of task a (in lines 19 and 22).

It is easy to understand why Nuijten made this mistake. The algorithm CalcLB for cumulative scheduling is derived from a similar algorithm for disjunctive scheduling (resources have capacity 1). In disjunctive scheduling, $C - c_i$ is always zero and $\text{rest}(\Theta, c_i)$ does not depend on d_Θ. It is thus always beneficial for a given r_Θ to add more tasks when computing the inner maximization. This is not the case in cumulative scheduling, where this dominance relation does not hold as the instance above indicates. We now prove formally that CalcLB does not compute $\overline{LB_2}(a)$ by tracing the algorithm.

Theorem 1 *Algorithm* CalcLB *does not compute* $\overline{LB_2}(i)$ $(i \in T)$.

Proof Consider the following instance on a resource of capacity 4:

task	r	d	p	c
a	0	69	4	1
b	1	2	1	4
c	0	3	1	2
d	0	3	1	2
e	2	3	1	1

We showed earlier that $LB_2(a) \geq 2$ by considering the pair (Ω, Θ) where $\Omega = T \setminus \{a\}$ and $\Theta = \{b\}$. Algorithm CalcLB considers only three due dates $\{2, 3, 69\}$ and performs the following processing.

[2] In CalcLB, $lct(t)$ corresponds to d_t, $est(t)$ to r_t, $a(t)$ to e_t, $sz(t)$ to $c(t)$, and $LB_{est}(t)$ to $\overline{LB_2}(t)$. Our notations are consistent with [BLPN01].

Algorithm 1 CalcLB

Require: X is an array of tasks sorted by non-decreasing release dates;
Require: Y is an array of tasks sorted by non-decreasing due dates;

```
1: for y ← 1 to n do
2:   if y = n ∨ d_{Y[y]} ≠ d_{Y[y+1]} then
3:     E ← 0; l ← −∞; for all c ∈ Sc do g_c ← −∞; endfor
4:     for i ← n downto 1 do
5:       if d_{X[i]} ≤ d_{Y[y]} then
6:         E ← E + e_{X[i]};
7:         if d_{X[i]} > l then l ← d_{X[i]}; endif
8:         for all c ∈ Sc do
9:           rest ← E − (l − r_{X[i]})(C − c);
10:          if rest/c > 0 then g_c ← max(g_c, r_{X[i]} + ⌈rest/c⌉);
11:        end for
12:      end if
13:      for all c ∈ Sc do G[i][c] ← g_c; endfor
14:    end for
15:    H ← −∞;
16:    for x ← 1 to n do
17:      if d_{X[x]} > d_{Y[y]} then
18:        if E + e_{X[x]} > (d_{Y[y]} − r_{X[x]}) × C then
19:          LB[x] ← max(LB[x], G[x][c_{X[x]}]);
20:        end if
21:        if H + (e_{X[x]}/C) > d_{Y[y]} then
22:          LB[x] ← max(LB[x], G[1][c_{X[x]}]);
23:        end if
24:      else
25:        H ← max(H, r_{X[x]} + E/C);
26:        E ← E − e_{X[x]};
27:      end if
28:    end for
29:  end if
30: end for
```

$\mathbf{d}_\Omega = 69$. All tasks have due dates not greater than 69, and the test $d_{X[x]} > d_{Y[y]}$ always fails in line 17. No bound is improved.

$\mathbf{d}_\Omega = 3$. The only task satisfying $d_{X[x]} > d_{Y[y]}$ (i.e., $d_i > d_\Omega$) is a, so only $\overline{LB_2}(a)$ can be updated. Since the tasks are considered by decreasing release dates starting with e, l is updated to 3 immediately and never decreases. As a consequence, *rest* is never positive and all values $G[t][c]$ are equal to $-\infty$ at the end of the first inner loop. No update can take place in the second inner loop.

$\mathbf{d}_\Omega = 2$. The only task with a due date not greater than 2 is b. Since $\alpha(\{b\}, i)$ does not hold for any task $i \neq b$, no bound is improved.

This shows that algorithm CalcLB does not improve any bound on this instance, contradicting the claim that CalcLB is an edge-finding algorithm. ■

Note that the proof shows an even stronger result: s_a will not be updated even by iterating CalcLB, since a fixpoint is reached after the first iteration.

The result directly propagates to the $O(n^2)$ algorithm NBLP (algorithm 8, section 3.3.3 in [BLPN01]). Indeed, NBLP refines the first inner loop of CalcLB and suffers from the same defect. (The same instance exhibits the mistake).[3]

It is also unlikely that the structure of CalcLB can be salvaged. Indeed, this would require the correct computation of all the G values in time $O(nk)$, which seems to be intrinsically two-dimensional. The algorithm proposed in this paper remedies this problem by removing the first inner loop and using dynamic programming to precompute the inner maximizations in the $LB_2(i)$ definitions. The dynamic programming algorithm exploits some new dominance rules, which are also used to simplify the second inner loop.

4 Dominance Properties

Before presenting the algorithm, it is important to review the dominance properties used by the algorithms.

4.1 Dominance Property for E-Feasibility

Testing E-feasibility only relies on a single dominance property based on the concept of task intervals [CL94].

Definition 3 (Task Intervals) *Let $L, U \in T$. The task interval Ω_L^U is the set of tasks*

$$\Omega_L^U = \{k \in T \mid r_k \geq r_L \ \wedge \ d_k \leq d_U\}.$$

Note that it is not always the case that $d_{\Omega_L^U} = d_U$ and $r_{\Omega_L^U} = r_L$. Indeed, the tasks L and U are not necessarily included in Ω_L^U. Algorithms for testing E-feasibility only need to consider task intervals.

Proposition 1 *E-feasibility testing only needs to consider task intervals.*

Proof Consider a set Ω such that $C(d_\Omega - r_\Omega) < e_\Omega$ and a set Ω_L^U such that $r_L = r_\Omega \wedge d_U = d_\Omega$. Since $\Omega \subseteq \Omega_L^U$, $C(d_{\Omega_L^U} - r_{\Omega_L^U}) < e_{\Omega_L^U}$. ■.

4.2 Dominance Properties for Edge Finding

Edge-finding algorithms heavily rely on dominance properties in order to reduce the pairs (Ω, Θ) to consider when updating a task i. This section reviews the dominance properties used in our algorithm. Some of them are well-known, others are new. The first three properties reduce the sets Ω that must be considered in the pairs (Ω, Θ) for a task i. The last two reduce the sets Θ to consider. In the following, we restrict attention to E-feasible CRPs only.

Definition 4 (Valid Pair) *A pair (Ω, Θ) is* valid *wrt task i if*

$$i \notin \Omega \ \wedge \ \alpha(\Omega, i) \ \wedge \ \Theta \subseteq \Omega \ \wedge \ \mathrm{rest}(\Theta, c_i) > 0.$$

[3] We will discuss NBLP again, once we have presented a correct edge-finding algorithm for cumulative resources.

Definition 5 (Maximal Pair) *A pair* (Ω, Θ) *is* maximal *wrt task* i *if it is valid and satisfies*

$$LB_2(i) = r_\Theta + \left\lceil \frac{1}{c_i} \operatorname{rest}(\Theta, c_i) \right\rceil .$$

Proposition 2 *The computation of* $LB_2(i)$ *for an E-feasible CRP only needs to consider pairs of the form* (Ω_L^U, Θ) $(L, U \in T)$.

Proof Consider a maximal pair (Ω, Θ) and a set Ω_L^U such that $r_L = r_\Omega \wedge d_U = d_\Omega$. Since $\Omega \subseteq \Omega_L^U$ and the inner maximization in $LB_2(i)$ only involves Θ, it suffices to prove that (Ω_L^U, Θ) is valid. Since $\alpha(\Omega, i)$ holds and the CRP is E-feasible, $i \notin \Omega_L^U$. Moreover, by definition of Ω_L^U and since $\Omega \subseteq \Omega_L^U$, $\alpha(\Omega_L^U, i)$ holds and the pair (Ω_L^U, Θ) is valid and maximal. ■

The following dominance property relates the pairs with task i.

Proposition 3 *The computation of* $LB_2(i)$ *for an E-feasible CRP may restrict attention to pairs* (Ω_L^U, Θ) *where* $d_U = d_{\Omega_L^U} < d_i$.

Proof Consider a maximal pair (Ω_L^U, Θ). There exists a task $U' \in \Omega_L^U$ such that $d_{U'} = d_{\Omega_L^U}$. Since $\Omega_L^U = \Omega_L^{U'}$, the pair $(\Omega_L^{U'}, \Theta)$ is also maximal. Assume now that $d_{U'} \geq d_i$ and let $\Omega' = \Omega_L^{U'} \cup \{i\}$. Since $d_{\Omega'} = d_{U'}$ and $\alpha(\Omega_L^{U'}, i)$ holds, it follows that $C\,(d_{\Omega'} - r_{\Omega'}) < e_{\Omega'}$, which contradicts E-feasibility. ■

Proposition 3 allows us to remove the constraint $i \notin \Omega$ from $LB_2(i)$, since it is implied by $d_U < d_i$. The following dominance property is new and imposes a restriction on the tasks L used to define the sets Ω_L^U for $LB_2(i)$.

Proposition 4 *The computation of* $LB_2(i)$ *for an E-feasible CRP only needs to consider pairs* (Ω_L^U, Θ) *where* $d_{\Omega_L^U} = d_U < d_i$ *and* $r_L = r_{\Omega_L^U \cup \{i\}}$.

Proof Consider a maximal pair (Ω_L^U, Θ) such that $d_U < d_i$ and let $L' \in T$ be a task such that $r_{L'} = \min(r_i, r_{\Omega_L^U})$. Since $\Omega_L^U \subseteq \Omega_{L'}^U$ and the inner maximization only depends on Θ, it suffices to show that $\Omega_{L'}^U$ is valid. Since $d_U < d_i$, $i \notin \Omega_{L'}^U$. Moreover, since $r_{\Omega_L^U \cup \{i\}} = r_{\Omega_{L'}^U \cup \{i\}}$ and $\Omega_L^U \subseteq \Omega_{L'}^U$, $\alpha(\Omega_{L'}^U, i)$ holds and the result follows. ■

The following proposition summarizes the first three dominance properties.

Proposition 5 *For a E-feasible CRP,* $LB_2(i)$ *may be computed as*

$$LB_2(i) = \max_{\substack{L, U \in T \\ \alpha(\Omega_L^U, i) \\ d_U = d_{\Omega_L^U} < d_i \\ r_L = r_{\Omega_L^U \cup \{i\}}}} \; \max_{\substack{\Theta \subseteq \Omega_L^U \\ \operatorname{rest}(\Theta, c_i) > 0}} r_\Theta + \left\lceil \frac{1}{c_i} \operatorname{rest}(\Theta, c_i) \right\rceil$$

The next two dominance properties concern the choice of Θ. The first one is the counterpart of Proposition 2 for Θ.

Proposition 6 *The computation of* $LB_2(i)$ *for an E-feasible instance only needs to consider pairs* (Ω_L^U, Ω_l^u) $(r_L \leq r_l \leq d_u \leq d_U)$ *satisfying* $r_l = r_{\Omega_l^u}$ *and* $d_u = d_{\Omega_l^u}$.

Proof Consider a maximal pair (Ω_L^U, Θ) and a set $\Omega_l^u \subseteq \Omega_L^U$ such that $r_l = r_\Theta \wedge d_u = d_\Theta$. It follows that $\Theta \subseteq \Omega_l^u$, $r_\Theta = r_{\Omega_l^u}$, and $\mathrm{rest}(\Theta, c_i) \leq \mathrm{rest}(\Omega_l^u, c_i)$. Hence, (Ω_L^U, Ω_l^u) is maximal for task i. ■

The above dominance properties restrict the set of pairs to consider in computing $LB_2(i)$. The next property is of a fundamentally different nature: it increases the set of pairs (Ω, Θ) to consider by relaxing the constraint $r_l \geq r_L$ (and thus $\Theta \subseteq \Omega$). This dominance relation, which generalizes Theorem 4.13 in [Nui94], enables us to amortize the precomputation of inner maximizations of $LB_2(i)$ $(i \in T)$ effectively and to simplify the second inner loop of CalcLB.

Proposition 7 *Consider the function LB_2' defined as*

$$LB_2'(i) = \begin{array}{c} \max \\ L, U \in T \\ \alpha(\Omega_L^U, i) \\ d_U = d_{\Omega_L^U} < d_i \\ r_L = r_{\Omega_L^U \cup \{i\}} \end{array} \quad \begin{array}{c} \max \\ l, u \in T \\ r_{\Omega_l^u} = r_l \\ d_{\Omega_l^u} = d_u \leq d_U \\ \mathrm{rest}(\Omega_l^u, c_i) > 0 \end{array} \quad r_l + \left\lceil \frac{1}{c_i} \mathrm{rest}(\Omega_l^u, c_i) \right\rceil$$

For any E-feasible CRP, $\overline{LB_2}(i) = \overline{LB_2'}(i)$, where $\overline{LB_2'}(i) = max(r_i, LB_2'(i))$.

Proof By Proposition 5 and Proposition 6, $LB_2(i)$ can be rewritten as

$$LB_2(i) = \begin{array}{c} \max \\ L, U \in T \\ \alpha(\Omega_L^U, i) \\ d_U = d_{\Omega_L^U} < d_i \\ r_L = r_{\Omega_L^U \cup \{i\}} \end{array} \quad \begin{array}{c} \max \\ l, u \in T \\ r_{\Omega_l^u} = r_l \\ d_{\Omega_l^u} = d_u \leq d_U \\ r_l \geq r_L \\ \mathrm{rest}(\Omega_l^u, c_i) > 0 \end{array} \quad r_l + \left\lceil \frac{1}{c_i} \mathrm{rest}(\Omega_l^u, c_i) \right\rceil$$

It follows that $LB_2(i) \leq LB_2'(i)$. Moreover, by definition of $\overline{LB_2}(i)$ and $\overline{LB_2'}(i)$, it is sufficient to consider the case where $LB_2'(i) > r_i$ and to show that $LB_2(i) \geq LB_2'(i)$. Consider $L, U, l, u \in T$ satisfying

$$\begin{cases} r_L = r_{\Omega_L^U \cup \{i\}} \\ \alpha(\Omega_L^U, i) \\ r_{\Omega_l^u} = r_l \\ d_{\Omega_l^u} = d_u \leq d_U = d_{\Omega_L^U} < d_i \\ \mathrm{rest}(\Omega_l^u, c_i) > 0 \\ LB_2'(i) = r_l + \left\lceil \frac{1}{c_i} \mathrm{rest}(\Omega_l^u, c_i) \right\rceil \end{cases}$$

If $r_l \geq r_L$, then $LB_2(i) \geq LB_2'(i)$. Otherwise, partition Ω_l^u in $\Theta \cup \Omega_L^u$ where $\Theta = \Omega_l^u \setminus \Omega_L^u$. The rest of the proof proceeds by a case analysis. Informally speaking, in the first case, the set Θ has enough energy to cover $C(r_L - r_l)$ and the computation of $LB_2(i)$ for (Ω_l^U, Ω_l^u) is at least as good as the computation of $LB_2'(i)$ on (Ω_L^U, Ω_l^u). In the second case, Θ does not cover $C(r_L - r_l)$ and $LB_2(i)$ on (Ω_L^U, Ω_L^u) is at least as good as $LB_2'(i)$.

Assumption 1: Consider the case

$$r_l + \frac{1}{c_i} \mathrm{rest}(\Omega_l^u, c_i) > r_L + \frac{1}{c_i} \mathrm{rest}(\Omega_L^u, c_i). \tag{1}$$

We first rewrite the left-hand side of (1). By definition of rest, we have

$$c_i r_l + \text{rest}(\Omega_l^u, c_i) = c_i r_l + e_{\Omega_l^u} - (C - c_i)(d_u - r_l) \tag{2}$$

since $d_{\Omega_l^u} = d_u$, and $r_{\Omega_l^u} = r_l$. We now handle the right-hand side of (1) and show that

$$\text{rest}(\Omega_L^u, c_i) \geq e_{\Omega_L^u} - (C - c_i)(d_u - r_L). \tag{3}$$

If $\Omega_L^u \neq \emptyset$, $\text{rest}(\Omega_L^u, c_i) = e_{\Omega_L^u} - (C - c_i)(d_{\Omega_L^u} - r_{\Omega_L^u})$ and the result follows since $d_{\Omega_L^u} \leq d_u$ and $r_{\Omega_L^u} \geq r_L$. If $\Omega_L^u = \emptyset$, $\text{rest}(\Omega_L^u, c_i) = 0$ by definition and $e_{\Omega_L^u} = 0$. To show (3), we must prove that $d_u > r_L$. The inequality (1) then becomes

$$c_i r_l + e_{\Omega_l^u} - (C - c_i)(d_u - r_l) > c_i r_L.$$

By E-feasibility of Ω_l^u, $C(d_u - r_l) \geq e_{\Omega_l^u}$. These two last inequalities show that

$$c_i r_l + c_i(d_u - r_l) > c_i r_L.$$

and thus $d_u > r_L$, establishing (3). We now show that

$$e_\Theta > C(r_L - r_l).$$

Rewriting (1) using (2) and (3) gives

$$\begin{aligned} c_i r_l + e_{\Omega_l^u} - (C - c_i)(d_u - r_l) &> c_i r_L + e_{\Omega_L^u} - (C - c_i)(d_u - r_L) \\ e_{\Omega_l^u} - e_{\Omega_L^u} - C d_u + C d_u &> (C - c_i)(r_L - r_l) + c_i(r_L - r_l) \\ e_\Theta &> C(r_L - r_l). \end{aligned}$$

Finally, it remains to show that $\alpha(\Omega_l^U, i)$ holds. Since $\Theta \cap \Omega_L^u = \emptyset$, $\Theta \subseteq \Omega_l^u$, and $d_u \leq d_U$, we have that $\Theta \cap \Omega_L^U = \emptyset$ and $\Theta \cup \Omega_L^U \subseteq \Omega_l^U$. Hence,

$$\begin{aligned} e_{\Omega_l^U} &= e_\Theta + e_{\Omega_L^U} \\ e_{\Omega_l^U} &> C(r_L - r_l) + e_{\Omega_L^U} \end{aligned}$$

and thus

$$C r_l + e_{\Omega_l^U} > C r_L + e_{\Omega_L^U}.$$

Since $\alpha(\Omega_l^U, i)$ holds, we have

$$\begin{array}{ll} C(d_{\Omega_L^U} - r_{\Omega_L^U \cup \{i\}}) < e_{\Omega_L^U \cup \{i\}} & \\ C(d_U - r_L) < e_{\Omega_L^U \cup \{i\}} & \text{since } U = d_{\Omega_L^U} \ \& \ r_L = r_{\Omega_L^U \cup \{i\}} \\ C(d_U - r_L) < e_{\Omega_L^U} + e_i & \text{since } d_U < d_i \\ C(d_U - r_l) < e_{\Omega_l^U} + e_i & \text{since } C r_l + e_{\Omega_l^U} > C r_L + e_{\Omega_L^U}. \end{array}$$

Since $d_U \geq d_{\Omega_l^U}$ and $r_l \leq r_{\Omega_l^U \cup \{i\}}$, it follows that

$$C(d_{\Omega_l^U} - r_{\Omega_l^U \cup \{i\}}) < e_{\Omega_l^U \cup \{i\}}$$

and $\alpha(\Omega_l^U, i)$ holds. As a consequence,

$$LB_2(i) \geq r_l + \frac{1}{c_i} \text{rest}(\Omega_l^u, c_i)$$

and the result $LB_2(i) \geq LB_2'(u)$ follows from the properties of ceil.

Assumption 2: Consider the case

$$r_l + \frac{1}{c_i} \text{rest}(\Omega_l^u, c_i) \leq r_L + \frac{1}{c_i} \text{rest}(\Omega_L^u, c_i).$$

If $\text{rest}(\Omega_L^u, c_i) \leq 0$, it follows directly that $r_l + \frac{1}{c_i} \text{rest}(\Omega_l^u, c_i) \leq r_L$ and thus that $LB_2'(i) \leq r_L$. But this contradicts our hypothesis $LB_2'(i) > r_i \geq r_L$. Hence $\text{rest}(\Omega_L^u, c_i) > 0$ and, since $\Omega_L^u \subseteq \Omega_L^U$,

$$LB_2(i) \geq r_L + \left\lceil \frac{1}{c_i} \text{rest}(\Omega_L^u, c_i) \right\rceil \geq LB_2'(i). \ \blacksquare$$

Algorithm 2 E-FEASIBILITY

Require: X is an array of tasks sorted by non-decreasing release dates;
Require: Y is an array of tasks sorted by non-decreasing due dates;
Ensure: returns true iff the instance is E-feasible;

```
1:  for y ← 1 to n do
2:    D ← d_{Y[y]}
3:    e ← 0
4:    for x ← n downto 1 do
5:      if d_{X[x]} ≤ D then
6:        e ← e + e_{X[x]}
7:        if C · (D − r_{X[x]}) < e then
8:          return false;
9:        end if
10:     end if
11:   end for
12: end for
13: return true;
```

5 Testing E-Feasibility

This section presents the standard algorithm for testing E-feasibility [Nui94]. The algorithm only considers task intervals and uses two arrays of tasks: an array X where the tasks are sorted by non-decreasing release dates and an array Y where the tasks are sorted by non-decreasing due dates. Because several tasks may have the same release dates or the same due dates, the algorithm works in fact with pseudo task intervals expressed in terms of the indices of the tasks in the arrays. More precisely, the pseudo task intervals are defined as

$$\widetilde{\Omega}_x^y = \left\{ X[j] \;\mid\; x \leq j \leq n \;\&\; d_{X[j]} \leq d_{Y[y]} \right\}$$

Note that $\widetilde{\Omega}_x^y \subseteq \Omega_{X[x]}^{Y[y]}$ and $\widetilde{\Omega}_x^y = \Omega_{X[x]}^{Y[y]}$when $x = 1$ or $r_{X[x]} > r_{X[x-1]}$. The key insight underlying the algorithm is to amortize the energy computation by using an inner-loop on the release dates, iterating down from the largest release date to the smallest release date. The algorithm is depicted in Algorithm 2 and its correctness follows from Proposition 1.

6 The Edge-Finding Algorithm

A simple use of the dominance relations leads to an $O(n^5)$ edge finder by exploring all tuples (i, L, U, l, u). However, the inner maximization of

$$r_\Theta + \left\lceil \frac{1}{c_i} \operatorname{rest}(\Theta, c_i) \right\rceil .$$

does not depend on Ω, except for the fact that $\Theta \subseteq \Omega$ or, more precisely, its relaxation $d_u \leq d_U$ due to Proposition 7. As a consequence, the loops on l and u may be outside the loops on L and U, reducing the runtime complexity. The new edge-finding algorithm is thus organized in two phases. The first phase uses dynamic programming to precompute the inner maximizations, while the

second phase computes the updates using the precomputed results. We start by presenting the precomputation.

6.1 The Precomputation

The precomputation performs the inner maximization in LB'_2, i.e.,

$$\max_{\substack{l,u \in T \\ d_u \leq d_U \\ \text{rest}(\Omega_l^u, c) > 0}} r_{\Omega_l^u} + \left\lceil \frac{1}{c} \text{rest}(\Omega_l^u, c) \right\rceil$$

for all $c \in Sc$ and $U \in T$. Once again, in practice, the algorithm works with pseudo task intervals and computes the values $R[c, y]$ defined as

$$R[c, y] = \max_{\substack{l,u \in T \\ d_u \leq d_{Y[y]} \\ \text{rest}(\Omega_l^u, c) > 0}} r_{\Omega_l^u} + \left\lceil \frac{1}{c} \text{rest}(\Omega_l^u, c) \right\rceil .$$

To obtain $R[c, y]$, the algorithm computes the values

$$RT[c, x, y] = \max_{\substack{x \leq x' \ \& \ y' \leq y \\ \text{rest}(\widetilde{\Omega}_{x'}^{y'}, c) > 0}} r_{x'} + \left\lceil \frac{1}{c} \text{rest}(\widetilde{\Omega}_{x'}^{y'}, c) \right\rceil .$$

and we have that $R[c, y] = RT[c, x, y]$. The RT values can be computed by the following recurrence relation.

Proposition 8 *Let* $RT[c, x, 0] = -\infty$ $(x \in N)$ *and* $RT[c, n+1, y] = -\infty$ $(y \in N)$. *For* $2 \leq x \leq n+1$ *and* $0 \leq y \leq n-1$, *we have*

$$RT[c, x-1, y+1] = \max \begin{cases} RT[c, x, y+1] \\ RT[c, x-1, y] \\ r_{X[x-1]} + \left\lceil \frac{1}{c} f\left(\text{rest}(\widetilde{\Omega}_{x-1}^{y+1}, c)\right) \right\rceil \end{cases}$$

where f *is defined by* $f(x) = x$ *if* $x > 0$ *and* $-\infty$ *otherwise.*

Proof The base cases correspond to empty sets and are valid. For the inductive case, consider x^* and y^* $(x \leq x^* \ \& \ y^* \leq y)$ such that

$$RT[c, x-1, y+1] = r_{x^*} + \left\lceil \frac{1}{c} \text{rest}(\widetilde{\Omega}_{x^*}^{y^*}, c) \right\rceil .$$

Either $x^* > x-1$ or $y^* < y$ or $x^* = x-1 \ \wedge \ y^* = y+1$. In the first two cases, $RT[c, x-1, y+1]$ is correct by induction. The third case is correct by definition of RT. ■

Algorithm 3 depicts a dynamic programming algorithm to compute the R values using the recurrence relation above. The algorithm, for a given c, computes the columns $RT[c, n, y], \ldots, RT[c, 1, y]$ in $O(n^2)$ time and $O(n)$ space. It dynamically computes the energy of task intervals instead of using an $O(n^2)$ array, which is the purpose of lines 8-9.

Algorithm 3 CalcR: Precomputation of the Bounds Updates in $O(n^2k)$ time

Require: X array of task sorted by non-decreasing release date
Require: Y array of task sorted by non-decreasing due date
Ensure: $R[c, y]$ is computed according to its specification

```
1: for all c ∈ Sc do
2:   for all y ∈ T do
3:     E[y] ← 0;
4:     R[c, y] ← −∞;
5:   end for
6:   for x ← n downto 1 do
7:     for y ← 1 to n do
8:       if d_{X[x]} ≤ d_{Y[y]} then
9:         E[y] ← E[y] + e_{X[x]};
10:      end if
11:      a ← R[c, y];
12:      b ← R[c, y − 1];
13:      rest ← E[y] − (C − c)(d_{Y_[y]} − r_{X[x]});
14:      c ←  if rest > 0 then r_{X[x]} + (1/c)⌈rest⌉ else −∞;
15:      R[c, y] ← max(a, b, c);
16:    end for
17:  end for
18: end for
```

Theorem 2 *Algorithm 3 is correct for E-feasible CRPs.*

Proof Direct consequence of Proposition 8. ■

6.2 The Edge Finding Algorithm

Once the precomputation is available, an $O(n^3)$ algorithm can be easily derived (see Algorithm 4). The key idea is to iterate over all Ls and Us in the definition of LB'_2, using the values $R[c, U]$ to update the bounds. The algorithm is a direct implementation of $\overline{LB'_2}$, with lines 7-12 computing the energy $E[x]$ of $\widetilde{\Omega}^{Y[y]}_{X[x]}$.

Theorem 3 *Algorithm 4 is correct for E-feasible CRPs.*

Proof Direct consequence of Theorem 2 and Proposition 7. ■

Algorithm CalcEFI can be improved by using an idea already present in CalcLB. Observe that line 17 in CalcEFI does not depend on x: only the condition in line 15 does. Hence the update in line 17 can be applied if there exists an x satisfying the condition in line 15 (provided that the condition in line 16 also holds) and we do not need to know x explicitly. As a consequence, the loop on x can be removed and replaced by an incremental computation of the condition in line 15 as the loop on i proceeds. More precisely, the idea of algorithm CalcEF, depicted in Algorithm 5, is to maintain the part of the condition which does not depend on i, i.e.,

$$ECF = \max_{x \leq i} \left(E[x] - C(d_{Y[y]} - r_{X[x]}) \right)$$

at each iteration of the loop.

Algorithm 4 CalcEFI: An Edge-Finder in $O(n^3)$ Time and $O(nk)$ Space

Require: X array of task sorted by non-increasing release date
Require: Y array of task sorted by non-decreasing due date
Ensure: $LB[i] = LB_2(X[i])$ $(1 \leq i \leq n)$

```
 1: R ← CalcR();
 2: for x ← 1 to n do
 3:    LB[x] ← r_{X[x]}
 4: end for
 5: for y ← 1 to n − 1 do
 6:    E ← 0;
 7:    for x ← n downto 1 do
 8:       if d_{X[x]} ≤ d_{Y[y]} then
 9:          E ← E + e_{X[x]};
10:       end if
11:       E[x] ← E;
12:    end for
13:    for x ← 1 to n do
14:       for i ← x to n do
15:          if E[x] + e_{X[i]} > C(d_{Y[y]} − r_{X[x]}) then
16:             if d_{X[i]} > d_{Y[y]} then
17:                LB[i] ← max(LB[i], R[c_{X[i]}, y])
18:             end if
19:          end if
20:       end for
21:    end for
22: end for
```

Theorem 4 *Algorithm 5 is correct for E-feasible CRPs.*

Proof Consequence of Theorem 3 and the fact that CalcEF maintains the invariant

$$ECF = \max_{x \leq i} \left(E[x] - C(d_{Y[y]} - r_{X[x]})\right)$$

after line 15. ■

6.3 Discussion

It is interesting to mention a couple of properties of CalcEF. The bottleneck of the algorithm is the computation of the R values which takes $O(n^2k)$ time. However, in practice, there is no need to precompute the entire array, since many values $R[c, y]$ may not be needed by the algorithm. A lazy implementation, which computes $R[c, y]$ on demand, runs in time $O(n^2 + \Delta n^2)$, where Δ is the number of distinct capacities required by the set of tasks whose bounds are updated. Worst-case improvements to the algorithm however require a way to compute the R values more efficiently.

The reader may also wonder if the "refinement" of NBLP over CalcLB would transpose to CalcEF. It appears however that NBLP uses another incorrect dominance rule in the computation of the first inner loop of algorithm CalcLB. Indeed, NBLP only considers those Θ that maximize $Cr_\Theta + e_\Theta$, which

Algorithm 5 CalcEF: An Edge-Finder in $O(n^2k)$ Time and $O(nk)$ Space

Require: X array of task sorted by non-increasing release date
Require: Y array of task sorted by non-decreasing due date
Ensure: $LB[i] = LB_2(X[i])$ $(1 \leq i \leq n)$

```
 1: R ← CalcR();
 2: for x ← 1 to n do
 3:    LB[x] ← r_{X[x]}
 4: end for
 5: for y ← 1 to n do
 6:    E ← 0;
 7:    for x ← n downto 1 do
 8:       if d_{X[x]} ≤ d_{Y[y]} then
 9:          E ← E + e_{X[x]};
10:       end if
11:       E[x] ← E;
12:    end for
13:    CEF ← −∞;
14:    for i ← 1 to n do
15:       CEF ← max(CEF, E[i] − C(d_{Y[y]} − r_{X[i]}));
16:       if CEF + e_{X[i]} > 0 then
17:          if d_{X[i]} > d_{Y[y]} then
18:             LB[i] ← max(LB[i], R[c_{X[i]}, y])
19:          end if
20:       end if
21:    end for
22: end for
```

is not valid. As a consequence, there exist instances for which CalcLB returns the correct lower bounds, but not NBLP. Consider the following instance with a resource of capacity 2 and tasks with capacity requirements equal to one.

task	r	d	p
a	0	69	51
b	1	5	4
c	4	6	2

NBLP does not make any update, although $LB_2(a) = 2$. Indeed, when $d_{Y[c]} = 6$ is considered, the release date d_a should be improved with respect to the set $\Omega = \Theta = \{b, c\}$. Instead of that, only $\Omega = \{b, c\}, \Theta = \{c\}$ is considered, due to the test of line 9 as $Cr_{\{b,c\}} + e_{\{b,c\}} = 8$ is smaller than $Cr_{\{c\}} + e_{\{c\}} = 10$.

7 Extended Edge Finding

This section considers the extended edge-finding rule from [Nui94]. Nuijten gives an $O(n^3k)$ algorithm for the extended edge finger and reference [BLPN01] claims the existence of an $O(n^3)$ algorithm but does not give the algorithm. This section proposes an extended edge-finding algorithm that runs $O(n^2k)$ time and $O(nk)$ space.

7.1 The Extended Edge-Finding Rule

Consider a set $\Omega \subseteq T$ and a task $i \in T \setminus \Omega$ such that $r_i \leq r_\Omega \leq r_i + p_i$. This new condition is interesting, since no tasks in Ω can be scheduled in $[r_i, r_\Omega)$. Under these conditions, Nuijten [Nui94] shows that if

$$C(d_\Omega - r_\Omega) < e_\Omega + (r_i + p_i - r_\Omega)c_i$$

then any feasible schedule satisfies

$$s_i \geq r_\Theta + \left\lceil \frac{1}{c_i} \operatorname{rest}(\Theta, c_i) \right\rceil$$

for all $\Theta \subseteq \Omega$ satisfying

$$\operatorname{rest}(\Theta, c_i) > 0.$$

The preconditions can be specified by the property $\beta(\Omega, i)$ defined as

$$\beta(\Omega, i) \iff \begin{cases} r_i \leq r_\Omega \leq r_i + p_i \\ C(d_\Omega - r_\Omega) < e_\Omega + (r_i + p_i - r_\Omega)c_i \end{cases}$$

The following proposition justifies why this rule is called the extended edge-finder.

Proposition 9 $r_i \leq r_\Omega \leq r_i + p_i \ \wedge \ \alpha(\Omega, i) \implies \beta(\Omega, i)$.

Proof Since $r_i \leq r_\Omega$, we have

$$C(d_\Omega - r_{\Omega \cup \{i\}}) = C(d_\Omega - r_\Omega) + C(r_\Omega - r_i).$$

Since $i \notin \Omega$, $e_{\Omega \cup \{i\}} = e_\Omega + p_i c_i$ and, since $\alpha(\Omega, i)$ holds,

$$C(d_\Omega - r_\Omega) + C(r_\Omega - r_i) < e_\Omega + p_i c_i.$$

Since $C \geq c_i$,

$$C(d_\Omega - r_\Omega) + c_i(r_\Omega - r_i) < e_\Omega + p_i c_i$$

and the result follows. ■

We now specify the extended edge-finder algorithm.

Specification 2 (Extended Edge-Finder) *An extended edge-finder is an algorithm which, given an E-feasible CRP, computes a vector*

$$\left\langle \overline{LB_4}(1), \ldots, \overline{LB_4}(n) \right\rangle$$

where

$$\overline{LB_4}(i) = \max(r_i, LB_2(i), LB_3(i))$$

and

$$LB_3(i) = \max_{\substack{\Omega \subseteq T \\ i \notin \Omega \\ \beta(\Omega, i)}} \ \max_{\substack{\Theta \subseteq \Omega \\ \operatorname{rest}(\Theta, c_i) > 0}} r_\Theta + \left\lceil \frac{1}{c_i} \operatorname{rest}(\Theta, c_i) \right\rceil$$

7.2 Dominance Properties

In general, the dominance properties of the extended edge finder are similar in nature to those of the standard edge finder. In the following, we focus on the differences and define *valid* pairs as before, except that the condition $\alpha(\Omega, i)$ is replaced by $\beta(\Omega, i)$. The first proposition simplifies the definition of $\beta(\Omega, i)$.

Proposition 10 *For any E-feasible CRP,*

$$\beta(\Omega, i) \iff \begin{cases} r_i \leq r_\Omega \\ C(d_\Omega - r_\Omega) < e_\Omega + (r_i + p_i - r_\Omega)c_i \end{cases}$$

Proof We only need to show that the right-hand side implies the left-hand side. If $r_\Omega > r_i + p_i$, then $e_\Omega + (r_i + p_i - r_\Omega)c_i \leq e_\Omega$. Thus $C(d_\Omega - r_\Omega) < e_\Omega$, which contradicts E-feasibility. ■

The following proposition restricts the sets of pairs (Ω, Θ) to consider. These are the same as in the standard case, except that $r_L = r_{\Omega_L^U}$ because of the nature of the extended rule.

Proposition 11 *The computation of $LB_3(i)$ for an E-feasible CRP only needs to consider pairs of the form (Ω_L^U, Ω_l^u) such that $r_L = r_{\Omega_L^U}$, $d_U = d_{\Omega_L^U}$, $d_u = d_{\Omega_l^u} \leq d_U < d_i$ and $r_l = r_{\Omega_l^u} \geq r_L$.*

Proof Similar to the proofs of Propositions 2, 3, and 4. ■

The following proposition is the counterpart of Proposition 7. It refers both to the standard and extended edge finders.

Proposition 12 *Let LB_3' be defined by*

$$LB_3'(i) = \max_{\substack{L, U \in T \\ \beta(\Omega_L^U, i) \\ d_U < d_i \\ r_L = r_{\Omega_L^U} \\ d_U = d_{\Omega_L^U}}} \max_{\substack{l, u \in T \\ r_l = r_{\Omega_l^u} \\ d_u = d_{\Omega_l^u} \leq d_U \\ \mathrm{rest}(\Omega_l^u, c_i) > 0}} r_l + \left\lceil \frac{1}{c_i} \mathrm{rest}(\Omega_l^u, c_i) \right\rceil$$

Then, for any E-feasible CRP, $LB_3'(i) \leq \max(r_i, LB_3(i), LB_2(i))$.

Proof The previous propositions claim that

$$LB_2(i) = \max_{\substack{L, U \in T \\ d_U < d_i \\ r_L = r_{\Omega_L^U} \\ d_U = d_{\Omega_L^U} \\ \beta(\Omega_L^U, i)}} \max_{\substack{l, u \in T \\ d_u = d_{\Omega_l^u} \leq d_U \\ r_l = r_{\Omega_l^u} \geq r_L \\ \mathrm{rest}(\Omega_l^u, c_i) > 0}} r_l + \left\lceil \frac{1}{c_i} \mathrm{rest}(\Omega_l^u, c_i) \right\rceil$$

It follows that $LB_3(i) \leq LB_3'(i)$. Moreover, it is sufficient to consider the case where $LB_3'(i) > r_i$ and to show that $\max(LB_2(i), LB_3) \geq LB_3'(i)$. Suppose that $LB_3'(i) > r_i$.

Let $L, U, l, u \in T$ satisfying:

$$\begin{cases} r_L = r_{\Omega_L^U} \\ d_{\Omega_L^U} = d_U < d_i \\ \beta(\Omega_L^U, i) \\ r_{\Omega_l^u} = r_l \\ d_u = d_{\Omega_l^u} \leq d_U \\ \text{rest}(\Omega_l^u, c_i) > 0 \\ LB_3'(i) = r_l + \left\lceil \frac{1}{c_i} \text{rest}(\Omega_l^u, c_i) \right\rceil \end{cases}$$

If $r_l \geq r_L$, (Ω_L^U, Ω_l^u) is a maximal valid pair and $LB_3(i) \geq LB_3'(i)$. Now suppose that $r_l < r_L$. As in Proposition 7, partition Ω_l^u in $\Theta \cup \Omega_L^u$, with $\Theta = \Omega_l^u \setminus \Omega_L^u$.

Assumption 1: Assume first that

$$r_l + \frac{1}{c_i} \text{rest}(\Omega_l^u, c_i) > r_L + \frac{1}{c_i} \text{rest}(\Omega_L^u, c_i)$$

which implies $e_\Theta > C(r_L - r_l)$. Now we have two cases.

case $\mathbf{r_l \geq r_i}$. We show that $LB_3(i) \geq LB_3'(i)$. Since $r_l = r_{\Omega_l^u}$, $d_u \leq d_U$, and $r_l < r_L$,

$$e_{\Omega_l^U} + c_i(r_i + p_i - r_{\Omega_l^U}) \geq e_{\Omega_l^U} + c_i(r_i + p_i - r_L).$$

Since $\Theta \cap \Omega_L^U = \emptyset$, $e_{\Omega_l^U} = e_\Theta + e_{\Omega_L^U}$ and

$$e_{\Omega_l^U} + c_i(r_i + p_i - r_{\Omega_l^U}) \geq e_\Theta + e_{\Omega_L^U} + c_i(r_i + p_i - r_L).$$

Since $\beta(\Omega_L^U, i)$ holds and $r_L = r_{\Omega_L^U}$, we have

$$e_{\Omega_l^U} + c_i(r_i + p_i - r_{\Omega_l^U}) > C(d_U - r_L) + e_\Theta$$

which implies by $e_\Theta > C(r_L - r_l)$ that

$$e_{\Omega_l^U} + c_i(r_i + p_i - r_{\Omega_l^U}) > C(d_U - r_L) + C(r_L - r_l).$$

Since $r_l = r_{\Omega_l^u}$ and $d_u \leq d_U$, we have $r_l = r_{\Omega_l^U}$ and thus

$$e_{\Omega_l^U} + c_i(r_i + p_i - r_{\Omega_l^U}) > C(d_U - r_{\Omega_l^U})$$

which implies $\beta(\Omega_l^U, i)$.

case $\mathbf{r_l < r_i}$. We show that $LB_2(i) \geq LB_3'(i)$. Since $e_{\Omega_l^U} = e_\Theta + e_{\Omega_L^U}$,

$$e_{\Omega_l^U} + e_i = e_\Theta + e_{\Omega_L^U} + e_i$$

and, since $e_\Theta > C(r_L - r_l)$, $\beta(\Omega_L^U, i)$ holds, and $e_i = p_i c_i$, we have

$$\begin{aligned} e_{\Omega_l^U} + e_i &> C(r_L - r_l) + C(d_U - r_L) - c_i(r_i + p_i - r_L) + p_i c_i \\ e_{\Omega_l^U} + e_i &> C(d_U - r_l) + c_i(r_L - r_i) \\ e_{\Omega_l^U} + e_i &> C(d_U - r_l) \end{aligned}$$

which implies $\alpha(\Omega_l^U, i)$.

Assumption 2: It remains to consider the case

$$r_l + \frac{1}{c_i} \text{rest}(\Omega_l^u, c_i) \leq r_L + \frac{1}{c_i} \text{rest}(\Omega_L^u, c_i),$$

which is similar to the same case in Proposition 7. ■

Corollary 1 *For any E-feasible CRP, we have*

$$\overline{LB_4}(i) = \max(r_i, LB_2'(i), LB_3'(i))$$

Algorithm 6 CalcEEFI: An Extended Edge-Finder in $O(n^3)$ Time.

Require: X array of task sorted by non-increasing release date
Require: Y array of task sorted by non-decreasing due date
Ensure: $LB[i] = \overline{LB}_4(X[i])$ $(1 \leq i \leq n)$
1: CalcEF();
2: **for** $y \leftarrow 1$ **to** $n-1$ **do**
3: $E \leftarrow 0$;
4: **for** $x \leftarrow n$ **downto** 1 **do**
5: **if** $d_{X[x]} \leq d_{Y[y]}$ **then**
6: $E \leftarrow E + e_{X[x]}$;
7: **end if**
8: $E[x] \leftarrow E$;
9: **end for**
10: **for** $x \leftarrow 1$ **to** n **do**
11: **for** $i \leftarrow 1$ **to** x **do**
12: **if** $E[x] + c_{X[i]}(r_{X[i]} + p_{X[i]} - r_{X[x]}) > C(d_{Y[y]} - r_{X[x]})$ **then**
13: **if** $d_{X[i]} > d_{Y[y]}$ **then**
14: $LB[i] \leftarrow \max(LB[i], R[c_{X[i]}, y])$
15: **end if**
16: **end if**
17: **end for**
18: **end for**
19: **end for**

7.3 The Extended Edge-Finding Algorithm

The extended edge-finding algorithm uses the same precomputation as the standard procedure, since the only change is the condition $\beta(\Omega, i)$ which replaces $\alpha(\Omega, i)$. Moreover, it is possible to derive an $O(n^3)$ algorithm CalcEEFI, which is essentially similar to CalcEFI. The only changes are the initialization of the LB values in line 1 by CalcEF, the loop on i that now goes from 1 to x and, of course, the condition $\beta(\Omega, i)$. CalcEEFI is shown in Algorithm 6.

Theorem 5 *Algorithm 6 is correct for E-feasible CRPs.*

Proof Direct consequence of Theorem 2 and Proposition 12. ■

The optimization to move from $O(n^3)$ to $O(n^2 k)$ is slightly more complex for the extended edge finder. Once again, observe that line 14 in CalcEEFI does not depend on x: only the condition in line 12 does. Moreover, the condition can be rewritten as

$$(C - c_{X[i]})r_{X[x]} + E[x] - Cd_{Y[y]} > -(c_{X[i]}(r_{X[i]} + p_{X[i]})).$$

It does not matter which x satisfies this test, only that there exists such a value. As a consequence, the algorithm precomputes the expression

$$CEEF[c, i] = \max_{x \geq i} ((C - c)r_{X[x]} + E[x] - Cd_{Y[y]}).$$

Observe that these expressions are precomputed for all capacities, since we do not know in advance the capacities of the tasks the test will be applied to.

Algorithm 7 CalcEEF: An Extended Edge-Finder in $O(n^2k)$ Time.

Require: X array of task sorted by non-increasing release date
Require: Y array of task sorted by non-decreasing due date
Ensure: $LB[i] = LB_4(X[i])$ $(1 \leq i \leq n)$

```
 1: CalcEF();
 2: for y ← 1 to n − 1 do
 3:    E ← 0;
 4:    for x ← n downto 1 do
 5:       if d_{X[x]} ≤ d_{Y[y]} then
 6:          E ← E + e_{X[x]};
 7:       end if
 8:       E[x] ← E;
 9:    end for
10:    for x ← 1 to n − 1 do
11:       if r_{X[x]} = r_{X[x+1]} then
12:          E[x + 1] ← E[x];
13:       end if
14:    end for
15:    for all c ∈ Sc do
16:       CEEF[c, n + 1] ← ∞;
17:    end for
18:    for x ← n downto 1 do
19:       for all c ∈ Sc do
20:          CEEF[c, x] ← max(CEEF[c, x + 1], (C − c)r_{X[x]} + E[x] − Cd_{Y[y]});
21:       end for
22:    end for
23:    for i ← 1 to n do
24:       if CEEF[c_{X[i]}, i] + c_{X[i]}(r_{X[i]} + p_{X[i]}) > 0 then
25:          if d_{X[i]} > d_{Y[y]} then
26:             LB[i] ← max(LB[i], R[c_{X[i]}, y])
27:          end if
28:       end if
29:    end for
30: end for
```

Hence it necessary to compute them prior to the loop instead of incrementally as in CalcEF. The resulting edge finder CalcEEF is shown in Algorithm 7. Observe lines 10-13 which establish the correspondence between $\widetilde{\Omega}_x^y$ and $\Omega_{X[x]}^{Y[y]}$ by ensuring that

$$E[x] = \max\{\ E[j]\ \mid\ r_{X[j]} = r_{X[x]}\}.$$

These lines are not necessary in CalcEF since its loops scan array X from 1 to n contrary to the loop in lines 18-20.

Theorem 6 *Algorithm 5 is correct for E-feasible CRPs.*

Proof Consequence of Theorem 5 and the correctness of the $CEEF[c, x]$ values which satisfy the specification

$$CEEF[c, i] = \max_{x \geq i} ((C - c)r_{X[x]} + E[x] - Cd_{Y[y]}). \blacksquare$$

8 Conclusion

This paper reconsidered edge-finding algorithms for cumulative scheduling. These algorithms are at the core of constraint-based schedulers and update the earliest starting dates and latest finishing dates of tasks that must be scheduled after or before a set of other tasks. The paper made three contributions. First, it indicated that Nuijten's algorithm, and its derivatives, are incomplete because they use an invalid dominance rule inherited from disjunctive scheduling. Second, the paper presented a novel edge-finding algorithm for cumulative resources which runs in time $O(n^2k)$, where n is the number of tasks and k the number of different capacity requirements of the tasks. The key design decision is to organize the algorithm in two phases: The first phase uses dynamic programming to precompute the innermost maximization in the edge-finder specification, while the second phase performs the updates based on the precomputation. Finally, the paper proposed the first extended edge-finding algorithms that run in time $O(n^2k)$, improving on the running time of existing algorithms.

References

[BLPN01] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling*. Kluwer Academic Publishers, 2001.

[CL94] Y. Caseau and F. Laburthe. Improving CLP Scheduling with Task Intervals. In *Proceedings of the 11th International Conference on Logic Programming (ICLP'94)*, pages 369–383, Santa Margherita Ligure, Italy, 1994.

[CP94] J. Carlier and E. Pinson. Adjustment of Heads and Tails for the Job-shop Problem. *European Journal of Operational Research*, 78:146–161, 1994.

[NA96] W. Nuijten and E. Aarts. A Computational Study of Constraint Satisfaction for Multiple Capacitated Job Shop Scheduling. *European Journal of Operational Research*, 90(2):269–284, 1996.

[Nui94] W. Nuijten. *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach*. PhD thesis, Eindhoven University of Technology, 1994.

[Vil04] Petr Vilim. O(n log n) Filtering Algorithms for Unary Resource Constraint. In *Proceedings of the First International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'04)*, pages 319–334, Nice, 2004.