Time-Table Disjunctive Reasoning for the Cumulative Constraint

Steven Gay, Renaud Hartert, Pierre Schaus

UCLouvain, ICTEAM,
Place Sainte Barbe 2,
1348 Louvain-la-Neuve, Belgium
{firstname.lastname}@uclouvain.be

Abstract. Scheduling has been a successful domain of application for constraint programming since its beginnings. The `cumulative` constraint – which enforces the usage of a limited resource by several tasks – is one of the core components that are surely responsible of this success. Unfortunately, ensuring bound-consistency for the `cumulative` constraint is already NP-Hard. Therefore, several relaxations were proposed to reduce domains in polynomial time such as Time-Tabling, Edge-Finding, Energetic Reasoning, and Not-First-Not-Last. Recently, Vilim introduced the Time-Table Edge-Finding reasoning which strengthens Edge-Finding by considering the time-table of the resource. We pursue the idea of exploiting the time-table to detect disjunctive pairs of tasks dynamically during the search. This new type of filtering – which we call *time-table disjunctive reasoning* – is not dominated by existing filtering rules. We propose a simple algorithm that implements this filtering rule with a $\mathcal{O}(n^2)$ time complexity (where n is the number of tasks) without relying on complex data structures. Our results on well known benchmarks highlight that using this new algorithm can substantially improve the solving process for some instances and only adds a marginally low computation overhead for the other ones.

Keywords: Constraint programming, Scheduling, Cumulative Constraint, Time-table, Disjunctive Reasoning.

1 Introduction

Many real-world scheduling problems involve cumulative resources. A resource can be seen as an abstraction of any renewable entity – as machinery, electricity, or even manpower – which is used to perform tasks (also called activities). Although many tasks could be scheduled simultaneously on a same resource, the total use of a resource cannot exceed a fixed capacity at any moment.

In this paper, we focus on a single cumulative resource with a discrete finite capacity $C \in \mathbb{N}$ and a set of n tasks $\mathcal{T} = \{1, \ldots, n\}$. Each task i has a starting time $s_i \in \mathbb{Z}$, a fixed duration $d_i \in \mathbb{N}$, and an ending time $e_i \in \mathbb{Z}$ such that the equality $s_i + d_i = e_i$ holds. Moreover, each task i consumes a fixed amount of

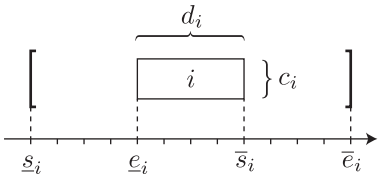

Fig. 1: Task i is characterized by its starting time s_i, its duration d_i, its ending time e_i, and its resource consumption c_i.

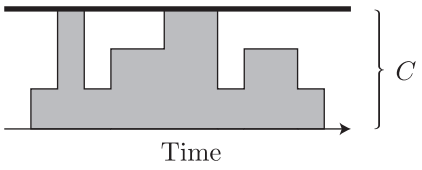

Fig. 2: Accumulated resource consumption over time. The `cumulative` constraint ensures that the maximum capacity C is not exceeded.

resource $c_i \in \mathbb{N}$ during its processing time. Tasks are non-preemptive, i.e., they cannot be interrupted during their processing time. In the following, we denote by $\underline{s}_i$ and $\overline{s}_i$ the earliest and the latest starting time of task i and by $\underline{e}_i$ and $\overline{e}_i$ the earliest and latest ending time of task i (see Fig. 1). The `cumulative` constraint [1] ensures that the accumulated resource consumption does not exceed the maximum capacity C at any time t (see Fig. 2):

$$\forall t \in \mathbb{Z} : \sum_{i \in \mathcal{T} : s_i \leq t < e_i} c_i \leq C. \tag{1}$$

Unfortunately, ensuring bound consistency for the `cumulative` constraint is already NP-Hard [11]. Therefore, many relaxations were proposed during the last two decades to remove inconsistent starting and ending times in polynomial time. Among them, the Time-Tabling filtering rule has been the subject of much research in the scheduling community [4,9,13]. The idempotent algorithm proposed by Letort in [9] implements Time-Tabling with a $\mathcal{O}(n^2)$ time complexity and has been successfully applied on problems with hundreds of thousands of tasks. The fastest (non-idempotent) known algorithm for Time-Tabling has a time complexity of $\mathcal{O}(n \log n)$ [13]. Despite its scalability, Time-Tabling suffers from limited filtering. On the other extreme, Energetic Reasoning [3,10] achieves a strong filtering at the cost of a prohibitive $\mathcal{O}(n^3)$ time complexity. Between these two extremes, several tradeoffs were proposed to balance strong filtering with low time complexity, e.g., Edge-Finding [6,15], Time-Table Edge-Finding [16], Time-Table Extended-Edge-Finding [13], or Not-First-Not-Last [14]. All the filtering rules listed above are subsumed by the filtering achieved by Energetic Reasoning at the exception of Not-First-Not-Last that is not comparable with Energetic Reasoning.

Surprisingly, Disjunctive Reasoning (DR) [3] has only been partially adapted to the cumulative context. In [2], Baptiste and Le Pape proposed to detect sets of tasks that cannot overlap in time without exceeding the amount of resource available initially. However, this approach is limited as it does not take in account the changes in the amount of resource available over time. This situation is illustrated in Fig. 3 where a task k has been fixed (by search or propagation). It is easy to see that tasks i and j cannot overlap in time due to the amount

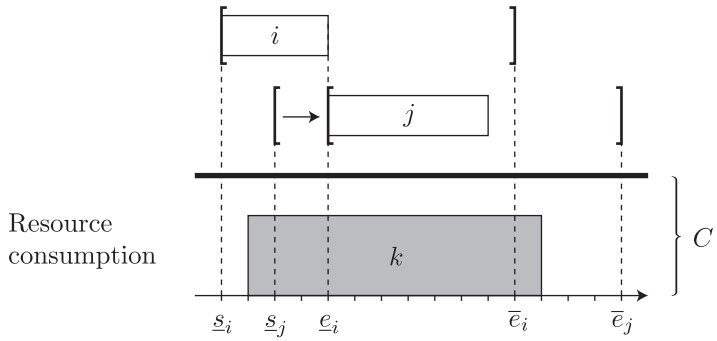

Fig. 3: Tasks i and j cannot overlap in time due to the amount of resource already consumed by task k. As j cannot be scheduled before i, j has to be scheduled after i: $e_i \leq s_j$. This situation is not detected by the approach proposed in [2].

of resource consumed by k. Unfortunately, this situation is not detected by the approach proposed by Baptiste and Le Pape.

In this work, we propose to improve Disjunctive Reasoning by considering changes in the amount of resource available. Similarly to the idea of Vilim in [16], we leverage the time-table – a core concept of Time-Tabling – to detect disjunctive pairs of tasks dynamically. Our new filtering rule – namely *Time-Table Disjunctive Reasoning* – is not subsumed by any known filtering rule. We propose a simple algorithm that implements this filtering rule with a $\mathcal{O}(n^2)$ time complexity without relying on complex data structures. We also propose two ways of improving the filtering of this algorithm. Our results on well known benchmarks from PSPLIB [7] highlight that Time-Tabling Disjunctive Reasoning is a promising filtering rule for state-of-the-art `cumulative` constraints. Indeed, using our algorithm can substantially improve the solving process of some instances and, at worst, only adds a marginally low computation overhead for the other ones.

This paper is structured as follows. Section 2 describes the time-table and the necessary background. Section 3 is dedicated to the Time-Table Disjunctive Reasoning rule and presents the algorithm and two possible extensions. The evaluation of our approach is presented in Section 4. This paper concludes on future works and possible improvements.

2 Mandatory Parts and Time-Table

Even tasks that are not fixed convey some information that can be used by filtering rules. For instance, tasks with a tight execution window must consume some resource during a specific time interval known as *mandatory part*.

Definition 1 (Mandatory part). *Let us consider a task $i \in \mathcal{T}$. The mandatory part of i is the time interval $[\overline{s}_i, \underline{e}_i[$. Task i has a mandatory part only if its latest starting time is smaller than its earliest ending time.*

If task i has a mandatory part, we know that task i will consume c_i of resource during all its mandatory part no matter its starting time. Fig. 4 illustrates the mandatory part of an arbitrary task i.

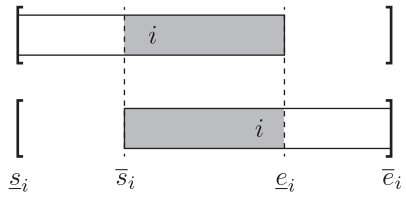

Fig. 4: Task i has a mandatory part $[\overline{s}_i, \underline{e}_i[$ if its latest starting time $\overline{s}_i$ is smaller than its earliest ending time $\underline{e}_i$: $\overline{s}_i < \underline{e}_i$. Task i always consumes the resource during its mandatory part no matter its starting time.

By aggregation, mandatory parts allow to have an optimistic view of the resource consumption over time. This aggregation is known as the *time-table* (also called minimum resource profile).

Definition 2 (Time-Table). *The time-table $TT_{\mathcal{T}}$ is the aggregation of the mandatory part of all the tasks in $\mathcal{T}$. It is formally defined as the following step function:*

$$TT_{\mathcal{T}} = t \in \mathbb{Z} \longrightarrow \sum_{i \in \mathcal{T} \mid \overline{s}_i \leq t < \underline{e}_i} c_i. \tag{2}$$

The problem is inconsistent if $\exists t \in \mathbb{Z} : \ TT_{\mathcal{T}}(t) > C$.

The time-table can be computed in $\mathcal{O}(n)$ with a sweep algorithm given the tasks sorted by latest starting time and earliest ending time [4,9,16].

3 Time-Table Disjunctive Reasoning

In order to explain Time-Table Disjunctive Reasoning, we will use some additional notations. Let I, J be time intervals. If $I \subseteq J$, we say that J *contains* I. If $I \cap J \neq \emptyset$, we say that I overlaps J, or that I and J overlap.

3.1 Disjunctive Reasoning and Minimum Overlapping Intervals

In [3], Baptiste et al. briefly describe Disjunctive Reasoning in the cumulative context as a reasoning on all pairs of tasks $i \neq j$ that enforces bound-consistency on the formula:

$$c_i + c_j \leq C \quad \vee \quad e_i \leq s_j \quad \vee \quad e_j \leq s_i. \tag{3}$$

The filtering rule to update start variables based on this formula is given next.

Proposition 1 (Disjunctive Reasoning). *Let us consider a pair of tasks $i \neq j$ in $\mathcal{T}$, such that $c_i + c_j > C$, $\underline{s}_j < \underline{e}_i$ and $\overline{s}_i < \underline{e}_j$. Then, $\underline{e}_i \leq s_j$ must hold.*

This rule states that if i and j cannot overlap, and if scheduling j at $\underline{s}_j$ would make it overlap i, then $e_i \leq s_j$, so the start of j must be at least $\underline{e}_i$. We say that task i is a *pushing* task while task j is a *pushed* task. A rule for filtering the ending times can be derived by symmetry.

The rule from Prop. 1 does some of the work of time-table filtering: when $c_i + c_j > C$ and i has a mandatory part, the reasoning on pair (i, j) is subsumed by time-tabling [3]. An additional filtering can be achieved by Disjunctive Reasoning when task i does not have a mandatory part. This filtering occurs when placing task j at $\underline{s}_j$ would make it overlap i in every schedule. It is based on the fact that j cannot contain a time interval that i must overlap. When i has no mandatory part, there is a minimum such interval.

Definition 3 (Minimum Overlapping Interval). *The minimum overlapping interval of task i, denoted* moi_i*, is the smallest time interval that overlaps i no matter the starting time of i. It is defined by the interval $[\underline{e}_i - 1, \overline{s}_i]$. Task i is always executed during at least one time point of* moi_i*.*

The moi of a task i is illustrated in Fig. 5.

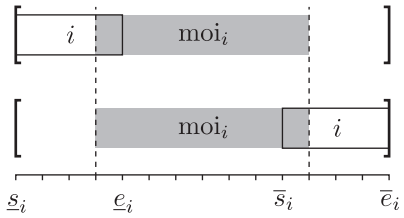

Fig. 5: Minimum overlapping interval of task i. Wherever i is placed, i must overlap moi_i, and moi_i is the smallest such interval.

When a task has a mandatory part, we consider that it has no minimum overlapping interval. Using the concept of minimum overlapping interval, it is possible to rewrite the part of Prop. 1 that is specific to Disjunctive Reasoning.

Proposition 2 (Restricted Disjunctive Reasoning). *Let us consider a pair of tasks $i \neq j$ such that task i has no mandatory part ($\underline{e}_i \leq \overline{s}_i$) and that $c_i + c_j > C$. If scheduling task j at its earliest starting time makes it completely overlap the minimum overlaping interval of i ($\text{moi}_i \subseteq [\underline{s}_j, \underline{e}_j[$), then $\underline{e}_i \leq s_j$ must hold.*

The rule from Prop. 2 is illustrated in Fig. 6. Algorithm 1 directly follows from this rule.

3.2 Restricted Time-Table Disjunctive Reasoning

One weakness of Disjunctive Reasoning lies in the fact that it does not take into account the changes in the amount of resource available over time (see Fig. 3).

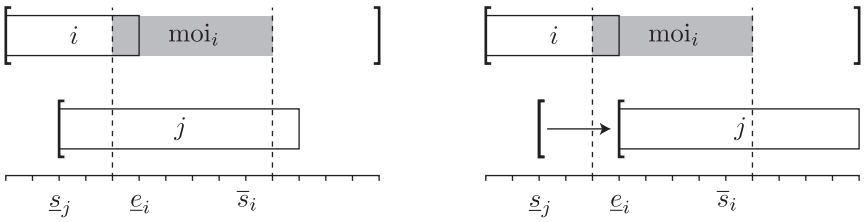

Fig. 6: On the left, tasks i and j cannot fit together for capacity reasons. Setting s_j to $\underline{s}_j$ would make j contain moi_i, and placing i would be impossible. On the right, the inconsistent values of s_j are removed, these are $t \leq \min(\text{moi}_i)$. This filtering is achieved by setting $\underline{s}_j$ to $\underline{e}_i$.

Algorithm 1: $O(n^2)$ algorithm to enforce rule of Prop. 2

Input: a set of tasks $\mathcal{T}$, capacity C
Input: an array s' mapping i to $\underline{s}_i$
Output: array s' mapping i to updated starting time

```
1 for i ∈ T such that e_i ≤ s̄_i do
2     for j ∈ T − {i} do
3         if c_i + c_j > C ∧ moi_i ⊆ [s_j, e_j[ then
4             s'_j ← max(s'_i, e_i)
```

In this section, we show how to exploit the information contained in the time-table (see Section 2) to propose an enhanced disjunctive filtering rule called *Time-Table Disjunctive Reasoning*.[1] We first introduce Time-Table Disjunctive Reasoning for the case where tasks i and j have no mandatory part and thus do not contribute to the time-table. This particular case saves us from removing the possible contribution of i or j from the time-table when applying a disjunctive reasoning. This restricting assumption will be relaxed to any pair of tasks later on in section 3.3.

Let us consider a pair of tasks $i \neq j$ with no mandatory parts such that $\text{moi}_i \subseteq [\underline{s}_j, \underline{e}_j[$. Then Prop. 2 only compares $c_i + c_j$ to C. However, tasks in $\mathcal{T} - \{i, j\}$ may not leave C units of resource available during the overlap of i and j. We derive a new rule that leverages the mandatory part of such tasks.

Proposition 3. *Let us consider a pair of tasks $i \neq j \in \mathcal{T}$ such that i and j have no mandatory part and that $c_i + c_j + \min_{t \in \text{moi}_i} \text{TT}_{\mathcal{T}}(t) > C$. If scheduling task j at its earliest starting time makes it completely overlap the minimum overlapping interval of i ($\text{moi}_i \subseteq [\underline{s}_j, \underline{e}_j[$), then $\underline{e}_i \leq s_j$ must hold.*

[1] The idea of leveraging the time-table to strengthen an existing filtering rule has already been applied successfully in [13,16].

Proof. If j contained moi_i, then j would increase consumption by c_j during all of moi_i, because j does not yet contribute to resource consumption. Then, placing i anywhere would increase consumption by c_i at some point t of moi_i, making consumption at t greater than C. Moreover, since $\text{moi}_i \subseteq [\underline{s}_j, \underline{e}_j[$, the duration of j is such that scheduling j before $\underline{e}_i$ makes j contain moi_i. Hence, these values are inconsistent, and $\underline{e}_i \leq s_j$ must hold. □

Using this rule, we can only filter values among tasks with no mandatory parts. Next section shows how to apply the same reasoning to every task.

3.3 Time-table Disjunctive Reasoning

Using the same idea as in [13,16], we strengthen our rule further by splitting every task in two parts, a *free* part and the mandatory part.

Definition 4 (Free part). *Let us consider a task $i \in \mathcal{T}$ such that i has a mandatory part ($\overline{s}_i < \underline{e}_i$). Its free part, denoted i_f, is a separate task with the same earliest starting time and latest ending time as i: $\underline{s}_{i_f} = \underline{s}_i$ and $\overline{e}_{i_f} = \overline{e}_i$. The duration of i_f is equal to the duration of i minus the size of its mandatory part: $d_{i_f} = d_i - (\underline{e}_i - \overline{s}_i)$. If i has no mandatory part, then $i = i_f$.*

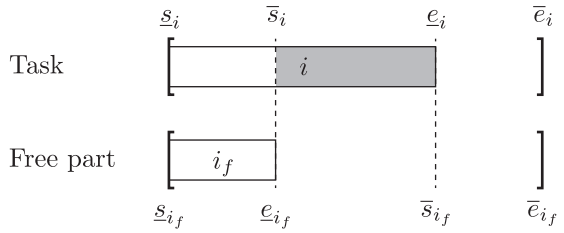

Fig. 7: A task i with a mandatory part and its free part i_f. The free part i_f always has a minimum overlapping interval moi_{i_f}.

Free parts have no mandatory part and always have an moi (see Fig. 7). In the remainder, we refer to $\mathcal{T}_f = \{i_f \mid i \in \mathcal{T} \wedge d_{i_f} > 0\}$ as the set of all the free parts of strictly positive duration (i.e. free parts of not assigned tasks).

Using free parts enables us to use any task in the reasoning, without worrying whether or not they contribute to the time-table. Notice that while the update is triggered by computations on free parts of tasks, the actual update should be made on tasks themselves, here s_j.

Proposition 4 (Time-Table Disjunctive Reasoning). *Let us consider a pair of tasks $i_f \neq j_f \in \mathcal{T}_f$ such that $c_i + c_j + \min_{t \in \text{moi}_{i_f}} \text{TT}_{\mathcal{T}}(t) > C$. If task j_f scheduled at its earliest starting time completely overlaps the minimum overlapping interval of i_f ($\text{moi}_{i_f} \subseteq [\underline{s}_{j_f}, \underline{e}_{j_f}[$), then, $\underline{e}_{i_f} \leq s_j$ must hold.*

Proof. Suppose that the premises are true, and then suppose $s_j \leq \min(\text{moi}_i)$. Since j_f contains moi_{i_f} when left-shifted and $d_j \geq d_{j_f}$, placing j before or at $\min(\text{moi}_{i_f})$ makes it contain moi_{i_f}. Notice that j does not contribute to the time-table during moi_{i_f}, since $\max(\text{moi}_{i_f}) \leq \underline{e}_{j_f} = \min(\underline{e}_j, \overline{s}_j)$.

If i has no mandatory part, it does not contribute to the time-table. This means that $\forall t \in \text{moi}_{i_f}, TT(t) = TT_{\mathcal{T}-\{i,j\}}(t)$. Then placing j before or at $\min(\text{moi}_i)$ increases resource consumption on moi_{i_f} by c_j, which prevents $i = i_f$ from being placed on its moi, and is contradictory.

If i has a mandatory part, it contributes to the time-table on $[\min(\text{moi}_{i_f}) + 1, \max(\text{moi}_{i_f}) - 1]$. Placing j before or at $\min(\text{moi}_{i_f})$ increases resource consumption at $\min(\text{moi}_{i_f})$ and at $\max(\text{moi}_{i_f})$ by c_j, This prevents i_f from being left-shifted or right-shifted, which in turn means that i itself cannot overlap these time points. Since it must overlap at least one of these points, this is contradictory. □

Algorithm 2 is an easy to implement $\mathcal{O}(n^2)$ algorithm combining moi and free parts abstractions. This algorithm enforces the updates of starting times given by Prop. 4. The tasks Pushing are candidate pusher tasks (taking the role of i). The tasks Pushed are candidate pushed tasks (take the role of j). For now, they are both $\mathcal{T}_f$. The time-table is basically an array of pairs (t, c) where t is a time and c is a consumption, it must be sorted by nondecreasing t. Its initialization in line 1 can be done in $\mathcal{O}(n \log n)$, by sorting tasks according to $\overline{s}$ and $\underline{e}$ and sweeping over these time points. In line 3, $consumption(i, \text{TT})$ can be implemented[2] as $\min_{t \in \text{moi}_{i_f}} \text{TT}(t)$, it can be computed in linear time on the time-table. Hence, this algorithm is $\mathcal{O}(n^2)$. Its correctness follows from Prop. 4.

Algorithm 2: Time-Table Disjunctive filtering algorithm.

Input: sets of tasks $\mathcal{T}$, Pushing $\subseteq \mathcal{T}_f$ and Pushed $\subseteq \mathcal{T}_f$, capacity C
Input: an array s' initially mapping i to $\underline{s}_i$
Output: array s' with updated starting times

```
1 TT ← initializeTimeTable(T)
2 for i_f ∈ Pushing do
3     gap ← C − c_i − consumption(i, TT)
4     for j_f ∈ Pushed −{i_f} do
5         if moi(i) ⊆ [s(j), e(j)[ then
6             if c_j > gap then
7                 s'_j ← max(s'_j, e(i))
```

Proposition 5. *The filtering of Time-Table Disjunctive Filtering is not subsumed by Energetic Reasoning nor by Not-First-Not-Last.*

[2] This primitive is voluntarily let abstract to describe further improvements.

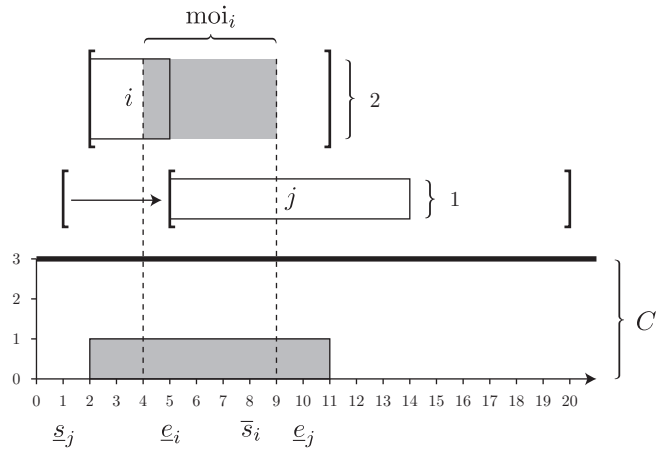

Fig. 8: Due to the time-table, tasks j and i cannot overlap. Since j cannot be scheduled before i, j has to be scheduled after i: $e_i \leq s_j$.

Proof. Figure 8 shows an example where Time-Table Disjunctive Reasoning can filter some values, but Energetic Reasoning and Not-First-Not-Last cannot. Task i is defined by $(\underline{s}_i, \overline{e}_i, d_i, c_i) = (2, 11, 3, 2)$, task j is defined by $(\underline{s}_j, \overline{e}_j, d_j, c_j) = (1, 20, 9, 1)$. Consumption in the resource could come from task k, defined by $(\underline{s}_k, \overline{e}_k, d_k, c_k) = (2, 11, 9, 1)$. Tasks i and j have no mandatory part, so $i_f = i$ and $j_f = j$. The condition $\text{moi}_i = [4, 8] \subseteq [\underline{s}_j, \underline{e}_j[= [1, 10[$ is satisfied, and the minimum of TT over $\text{moi}_i = [4, 8]$ is 1. It means that the two tasks i and j, consuming respectively 2 and 1 unit of resource, are not allowed to overlap over $[4, 8]$. Hence $\underline{s}_j$ is updated to $\min(\text{moi}_i) + 1 = \underline{e}_i = 5$. □

3.4 Improvements

The computation of allowed gap in Prop. 4, reflected at line 3 of Algorithm 2, can be strengthened in some cases, allowing to filter more values.

Pushing task does not fit inside its moi. When the duration of i_f is larger than its moi, we can strengthen the gap allowed by i by taking the minimum of the time-table on extremities of moi_{i_f} instead of taking it on the whole interval:

Proposition 6 (Improvement 1). *Let $i_f \neq j_f \in \mathcal{T}_f$, such that $|\,\text{moi}_{i_f}\,| - 1 \leq d_{i_f}$. Suppose $\text{moi}_{i_f} \subseteq [\underline{s}_{j_f}, \underline{e}_{j_f}[$ and*

$$c_{i_f} + c_{j_f} + \min(\text{TT}(\min(\text{moi}_{i_f})), \text{TT}(\max(\text{moi}_{i_f}))) > C$$

Then $\underline{e}_{i_f} < s_j$ must hold.

Proof. If $s_j \leq \min(moi_{i_f})$, then j contains moi_{i_f}, so it contains the extremities of moi_{i_f}. Thus, j makes the consumption at $\min(\text{moi}_{i_f})$ and $\max(\text{moi}_{i_f})$ increase by c_j. Because of its duration, i_f must overlap at least one of $\min(\text{moi}_{i_f})$ and $\max(\text{moi}_{i_f})$. Doing so would overload the resource, so this is contradictory. □

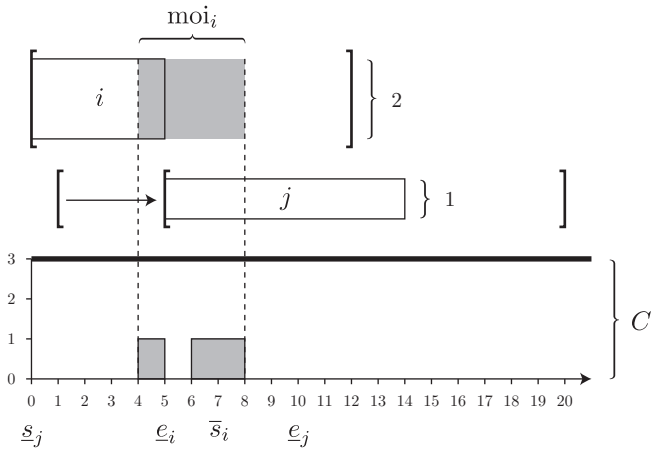

Fig. 9: Illustration of Prop. 6

Example 1. We refer to the Figure 9. Task $i = i_f$ has $\text{moi}_i = [4, 7]$, and $3 = |\,\text{moi}_i\,| - 1 \leq d_i = 5$. Prop. 4 does not cause any update, since $\min_{t \in \text{moi}_i} \text{TT}(t) = 0$. However, Prop. 6 would trigger and would compute a smaller gap, since $\min(\text{TT}(4), \text{TT}(7)) = 1$. Hence, Prop. 6 allows us to adjust the starting time of j to 5.

Notice that improving Algorithm 2 using Prop. 6 can be done by changing consumption(i, TT) at line 3 to compute the minimum only at the extremities of moi_i when $|\,\text{moi}_{i_f}\,| \leq d_{i_f} + 1$.

Pushing task has a mandatory part When task i has a mandatory part, despite i_f's domain, the consumption of i_f will not really be scheduled inside moi_{i_f}. Thus, we can strengthen the gap in the same way as the previous improvement:

Proposition 7 (Improvement 2). *Let $i_f \neq j_f \in \mathcal{T}_f$, such that i has a mandatory part. Suppose* $\text{moi}_{i_f} \subseteq [\underline{s}_{j_f}, \underline{e}_{j_f}[$ *and*

$$c_{i_f} + c_{j_f} + \min(\text{TT}(\min(\text{moi}_{i_f})), \text{TT}(\max(\text{moi}_{i_f}))) > C$$

Then $\underline{e}_{i_f} < s_j$ must hold.

Proof. The argument is the same as for Prop. 6: i must overlap either $\min(\text{moi}_{i_f})$ or $\max(\text{moi}_{i_f})$, hence computing the gap using only these points is sufficient. □

Example 2. We refer to the Figure 10. Task i has a mandatory part. Since $4 = |\,\text{moi}_{i_f}\,| - 1 > d_{i_f} = 2$, we are not in the case of application of Prop. 6. We cannot apply Prop. 4 either, since $\min_{t \in \text{moi}_{i_f}}(\text{TT}(t))$ is 1, allowing i and j to overlap. Nevertheless, since i has a mandatory part, we can apply Prop. 7 and use $\min(\text{TT}(4), \text{TT}(8)) = 2$ to compute the gap, which forbids i and j to overlap in moi_{i_f}. Hence, Prop.7 allows us to adjust the starting time of j to 5.

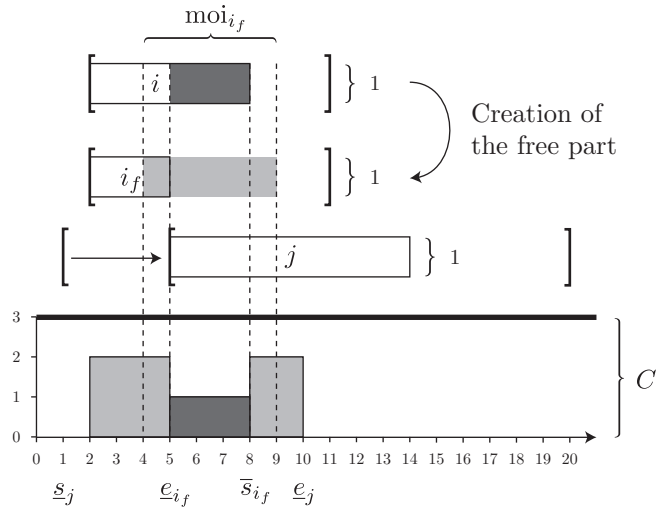

Fig. 10: Illustration of Prop. 7

Once more, improving Algorithm 2 using Prop. 7 can be done by changing consumption(i, TT) at line 3 to compute the minimum only at the extremities of moi$_i$ when i has a mandatory part.

Reducing the number of pairs to consider Our algorithm has a theoretical $\mathcal{O}(n^2)$ time complexity, but it is possible to reduce it in practice by removing tasks that cannot push, and by removing tasks that cannot be pushed. Refining these sets allows to keep the same filtering, while examining much less than the theoretical $\mathcal{O}(n^2)$ pairs of tasks.

First, pushing tasks must have a tight enough moi and high enough consumption during their moi to push any other task. After refining Pushing, we found useful to refine tasks of Pushed according to their time location: a pushed task must not be schedulable before the minimal earliest start of pushing tasks mois, and must be updatable at least by their maximal earliest end.

This procedure is formalized in Algorithm 3.

4 Experiments and Results

This section presents the experimental evaluation of Time-Table Disjunctive Reasoning (TTDR) on several well known benchmarks of the Resource Constraints Project Scheduling Problem (RCPSP) from PSPLIB [7]. The aim of these experiments is to measure the performance of the `cumulative` constraint when TTDR is added to a set of filtering rules. To achieve this, we compared the performance of classical filtering rules with and without TTDR. Here is the exhaustive list of the compared algorithms:

– Time-Tabling (TT) is implemented using a fast variant of [9];

Algorithm 3: Refinement of Pushing and Pushed with $\mathcal{O}(n)$ overhead

Input: a set of tasks $\mathcal{T}$, initialized TT, capacity C
Output: Pushing and Pushed arrays for TTDR input

$D = \max d_{i_f}$, $G = \max c_{i_f}$
$\text{Pushing}_0 = \{i_f \in \mathcal{T}_f \text{ s.t. } |\text{moi}_{i_f}| \leq D\}$
for $i_f \in \text{Pushing}_0$ **do**
    $\text{gap}_{i_f} \leftarrow C - c_i - \text{consumption}(i, \text{TT})$
$\text{Pushing} = \{i_f \in \text{Pushing}_0 \text{ s.t. } \text{gap}_{i_f} < G\}$
$S = \min_{i_f \in \text{Pushing}} \overline{s}_{i_f}$, $E = \max_{i_f \in \text{Pushing}} \underline{e}_{i_f}$
$\text{Pushed} = \{i_f \in \mathcal{T}_f \text{ s.t. } \underline{s}_{j_f} < E \wedge S < \underline{e}_{j_f}\}$

- Time-Table Disjunctive Reasoning (TTDR) implemented as described in this paper with all proposed improvements;
- Edge-Finding (EF) is implemented as proposed in [6];
- Energetic Reasoning (ER) is the well known implementation proposed by Baptiste et al. in [3]. We added some improvements proposed by Derrien et Petit to reduce the number of considered intervals [5];
- Not-First-Not-Last (NFNL) is implemented with a $\mathcal{O}(n^3)$ variant of the algorithm proposed by Schutt et al. [14].

All the algorithms were implemented in the open-source OscaR Solver [12].The priorities chosen for cumulative constraints in the propagation queue are such that TT is executed first, then TTDR, EF, ER and finally NFNL. We used a classic SetTimes search [8], breaking ties by taking a task of minimal duration.

We used a machine with a 4-core, 8 thread Core(TM) i7-2600 CPU @ 3.40GHz processor and 8GB of RAM under GNU/Linux. using Java SE 1.7 JVM.

We report the cumulated distribution $F(\tau)$ of instances solved within computational limit τ in Fig. 11 and Fig. 13. On the left column, τ refers to time, on the right it refers to the size of the search tree; the x-axis is logarithmic in both cases. $F(\tau) = k$ means that k instances where solved under τ ms or nodes. We set a timeout of 90s for every computation.[3] Due to a lack of room, we only display results obtained on instances with 30 and 120 tasks. Results obtained on instances with 60 tasks are similar. However, we observed that adding TTDR has a very little effect on instances with 90 tasks in which only two additional instances were closed using TTDR.

In Fig. 12 and Fig. 14, we report results for destructive lower bound experiments, that compute the best lower bound given by propagation alone.

Despite its $\mathcal{O}(n^2)$ theoretical complexity, the algorithm for TTDR is more of a lightweight algorithm. The only computation overhead appears on very small time limits when TTDR is used with TT. However, the additional filtering of TTDR quickly takes over, allowing to solve more instances for a given time

[3] Which is why time graphs will not show points further than 2^{17}ms.

limit τ. The PSPLIB instances are well-known to be rather disjunctive than cumulative.[4] Adding energy-based reasoning on PSPLIB instances is a risky trade-off. Indeed, energy-based reasoning does not trivialize much the PSPLIB instances, whereas TTDR does improve on the number of instances solved by TT alone. This confirms experimentally that TTDR is complementary to existing energy-based filtering for the `cumulative` constraint (see Prop. 5)

Finally, we also see that the gain in solved instances does not drop from the 30-task set to the 120-task set. This means that the filtering depends more on the nature of instances than on their size, and that the $\mathcal{O}(n^2)$ algorithm for TTDR scales well.

5 Conclusion

This paper introduces Time-Tabling Disjunctive Reasoning – a new filtering rule that leverages the time-table to detect disjunctive pairs of tasks dynamically. By relying on minimum overlapping intervals, this filtering rules achieve a new type of filtering that is not subsumed by existing filtering rules such as Energetic Reasoning or Not-First-Not-Last. Besides its novelty, Time-Table Disjunctive Reasoning can be implemented with a simple $\mathcal{O}(n^2)$ algorithm that does not rely on complex data-structures. Benefits of using Time-Table Disjunctive Reasoning in combination with other filtering rules were evaluated on well-known benchmarks from PSPLIB. Our results highlight that Time-Table Disjunctive Reasoning is a promising filtering rule to extend state-of-the-art `cumulative` constraints. Indeed, using our algorithm can substantially improve the solving process of some instances and, at worst, only adds a marginally low overhead for the other ones.

Although the strengthening proposed by Time-Table Disjunctive Reasoning is already a good tradeoff in terms of speed and filtering, it could be improved further. For instance, its practical complexity could be reduced by using sweeping techniques to prevent the examination of non-overlapping pairs of tasks. An even more interesting improvement would be on the filtering side. For instance, one may be able to strengthen the filtering by considering minimum overlapping intervals on more than one task at the same time.

Acknowledgments. Steven Gay is financed by project Innoviris 13-R-50 of the Brussels-Capital region. Renaud Hartert is a Research Fellow of the Fonds de la Recherche Scientifiques - FNRS. The authors would like to thank Sascha Van Cauwelaert for sharing his instances and parser, and the reviewers for suggestions of experiments.

[4] A problem is said to be highly disjunctive when many pairs of activities cannot execute in parallel; on the contrary, a problem is highly cumulative if many activities can effectively execute in parallel [2].

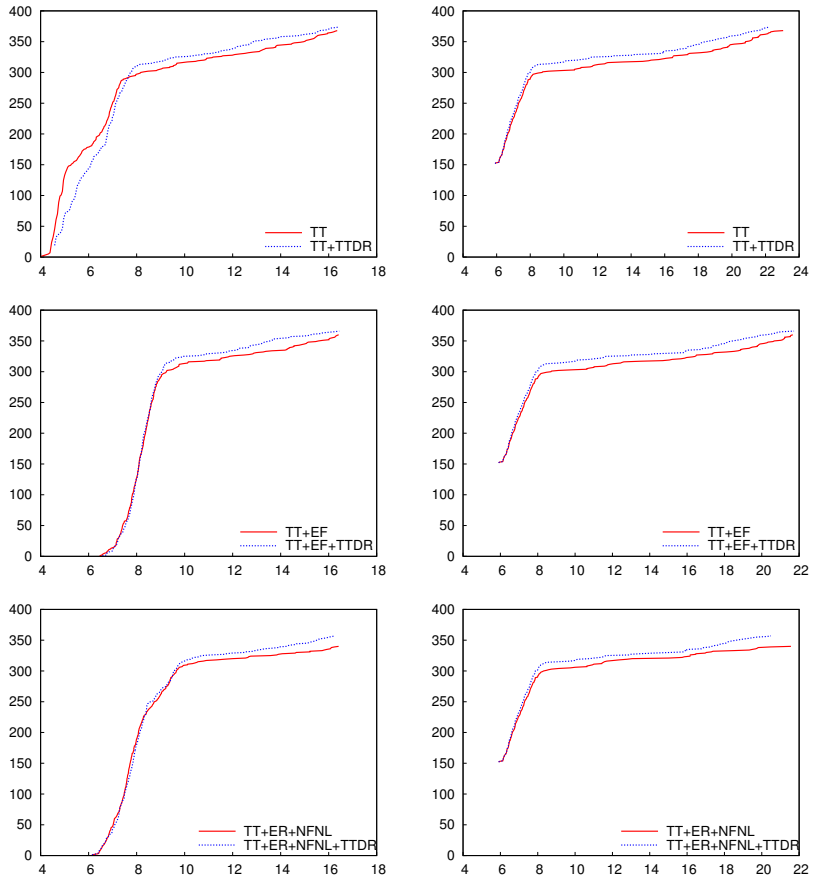

Fig. 11: Plots for all instances of PSPLIB30. y-axis is the cumulative number of solved instances. In the left column, x-axis is $log_2(t)$, with t time in ms. In the right column, x-axis is $log_2(n)$, with n the number of nodes to find optimal and prove optimality.

Stack	TT	TT+EF	TT+ER+NFNL
Score	26364	26712	26765
+TTDR, Score	26543	26815	26845
+TTDR, #Improvements	104	73	65

Fig. 12: Results for destructive lower bound experiments. Score is the sum of proven lower bounds, with the original stack or with TTDR. #Improvements shows the number of instances where adding TTDR gives a strictly higher bound.

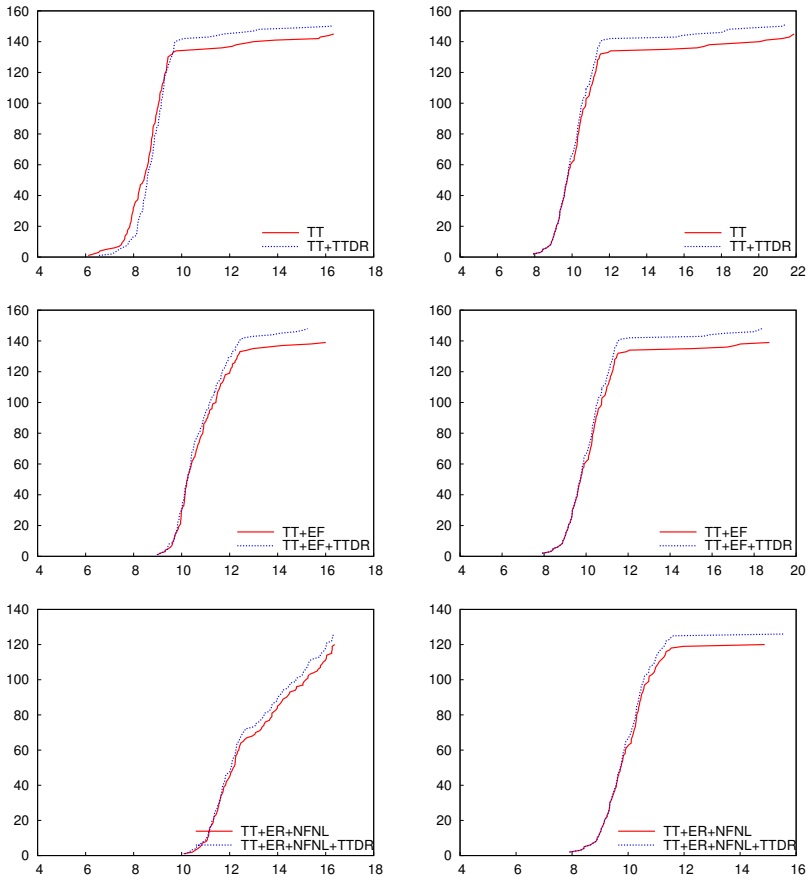

Fig. 13: Plots for all instances of PSPLIB120. y-axis is the cumulative number of solved instances. In the left column, x-axis is $log_2(t)$, with t time in ms. In the right column, x-axis is $log_2(n)$, with n the number of nodes to find optimal and prove optimality.

Stack	TT	TT+EF	TT+ER+NFNL
Score	58365	69074	69509
+TTDR, Score	58575	69117	69536
+TTDR, #Improvements	132	33	22

Fig. 14: Results for destructive lower bound experiments. Score is the sum of proven lower bounds, with the original stack or with TTDR. #Improvements shows the number of instances where adding TTDR gives a strictly higher bound.

References

1. Abderrahmane Aggoun and Nicolas Beldiceanu. Extending chip in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7):57–73, 1993.
2. Philippe Baptiste and Claude Le Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints*, 5(1-2):119–139, 2000.
3. Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer, 2001.
4. Nicolas Beldiceanu and Mats Carlsson. A new multi-resource cumulatives constraint with negative heights. In *CP*, Lecture Notes in Computer Science, pages 63–79, 2002.
5. Alban Derrien and Thierry Petit. A new characterization of relevant intervals for energetic reasoning. In *Principles and Practice of Constraint Programming*, pages 289–297. Springer, 2014.
6. Roger Kameugne, Laure Pauline Fotso, Joseph Scott, and Youcheu Ngo-Kateu. A quadratic edge-finding filtering algorithm for cumulative resource constraints. *Constraints*, 19(3):243–269, 2014.
7. Rainer Kolisch, Christoph Schwindt, and Arno Sprecher. Benchmark instances for project scheduling problems. In *Project Scheduling*, pages 197–212. Springer, 1999.
8. Claude Le Pape, Philippe Couronné, Didier Vergamini, and Vincent Gosselin. Time-versus-capacity compromises in project scheduling, 1994.
9. Arnaud Letort, Nicolas Beldiceanu, and Mats Carlsson. A scalable sweep algorithm for the cumulative constraint. In *Principles and Practice of Constraint Programming*, pages 439–454. Springer, 2012.
10. Pierre Lopez, Jacques Erschler, and Patrick Esquirol. Ordonnancement de tâches sous contraintes: une approche énergétique. *Automatique-productique informatique industrielle*, 26(5-6):453–481, 1992.
11. Wilhelmus Petronella Maria Nuijten. *Time and resource constrained scheduling: a constraint satisfaction approach.* PhD thesis, Technische Universiteit Eindhoven, 1994.
12. OscaR Team. OscaR: Scala in OR, 2012. Available from `https://bitbucket.org/oscarlib/oscar`.
13. Pierre Ouellet and Claude-Guy Quimper. Time-table extended-edge-finding for the cumulative constraint. In *Principles and Practice of Constraint Programming*, pages 562–577. Springer, 2013.
14. Andreas Schutt and Armin Wolf. A new O(n^2 log n) not-first/not-last pruning algorithm for cumulative resource constraints. In *Principles and Practice of Constraint Programming–CP 2010*, pages 445–459. Springer, 2010.
15. Petr Vilím. Edge finding filtering algorithm for discrete cumulative resources in O(kn log n). In *Principles and Practice of Constraint Programming-CP 2009*, pages 802–816. Springer, 2009.
16. Petr Vilím. Timetable edge finding filtering algorithm for discrete cumulative resources. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 230–245. Springer, 2011.