Raisonnement disjonctif pour la Contrainte `Cumulative` *

Steven Gay Renaud Hartert Pierre Schaus

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

Résumé

L'ordonnancement est un domaine dans lequel la programmation par contraintes a pu rapidement s'illustrer. Particulièrement, la contrainte `cumulative` – chargée d'assurer la cohérence de l'utilisation limitée d'une ressource par un ensemble de tâches – est l'un des composant critique probablement responsable de ce succès. Malheureusement, assurer la cohérence-de-bornes pour cette contrainte est déjà un problème NP-Difficile. Dès lors, de nombreuses relaxations furent proposées afin de réduire les domaines en des temps polynomiaux parmi lesquelles : Time-Tabling, Edge-Finding, Raisonnement Énergétique et Not-First-Not-Last. Petr Vilim a récemment introduit les raisonnements de type Time-Table Edge-Finding qui renforcent le raisonnement Edge-Finding en considérant le profil obligatoire de la ressource. Nous poursuivons l'idée d'exploiter ce même profil afin de détecter des paires de tâches disjointes de façon dynamique durant la recherche. Ce nouveau type de raisonnement – que nous appelons Time-Table Disjunctive Reasoning – n'est dominé par aucun raisonnement actuellement publié. Nous proposons un algorithme simple, ne se basant sur aucune structure de données complexe, qui implémente une procédure de filtrage basée sur ce raisonnement avec une complexité temporelle de $\mathcal{O}(n^2)$ où n correspond au nombre de tâches. Nos résultats sur des instances connues mettent en exergue les apports de notre nouveau propagateur sur certaines instances et son surcoût dérisoire sur les autres instances.

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.

1 Introduction

De nombreux problèmes réels d'ordonnancement requièrent l'utilisation de ressources cumulatives. Une ressource peut être vue comme une abstraction pour n'importe quelle entité renouvelable – telle que des machines, de l'électricité ou encore de la force de travail – utilisée afin d'exécuter des tâches (aussi appelées activités). Bien que de nombreuses tâches peuvent être exécutées en parallèle sur une même ressource, l'utilisation totale de la ressource à un instant donné est limitée par une capacité fixe. Dans cet article, nous nous concentrons sur une seule ressource cumulative ayant pour capacité discrète $C \in \mathbb{N}$ et un ensemble de tâches $\mathcal{T} = \{1, \ldots, n\}$. Chaque tâche i est caracté-

*Traduction de l'article CPAIOR 2015 [7]

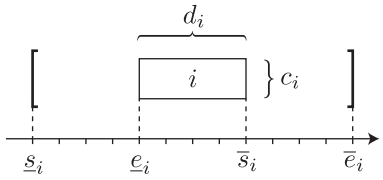

FIGURE 1 – Une tâche i caractérisée par sa date de départ s_i, sa durée d_i, sa fin e_i, et sa consommation de ressource c_i.

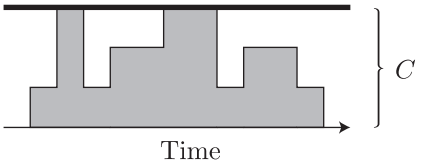

FIGURE 2 – Consommation de ressource cumulée au cours du temps. La contrainte `cumulative` s'assure que la capacité maximale C n'est pas dépassée.

risée par un temps de départ $s_i \in \mathbb{Z}$, une durée fixe $d_i \in \mathbb{Z}$ et un temps de fin $e_i \in \mathbb{Z}$ de manière à ce que l'égalité $s_i + d_i = e_i$ soit vérifiée. Aussi, chaque tâche i consomme une certaine quantité fixe de ressource $c_i \in \mathbb{N}$ pendant son temps d'exécution. Notons que les tâches ne peuvent pas être morcelées pendant leur temps d'exécution. Par la suite, nous dénotons par $\underline{s}_i$ et $\overline{s}_i$ la date de départ au plus tôt et la date de départ au plus tard et par $\underline{e}_i$ et $\overline{e}_i$ la date de fin au plus tôt et la date de fin au plus tard de la tâche i (voir FIGURE 1). La contrainte `cumulative` [1] assure que l'utilisation totale de la ressource ne dépasse jamais sa capacité C et ce pour tout temps t (voir FIGURE 2) :

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

Malheureusement, assurer ne serait-ce que la cohérence-de-bornes est déjà un problème NP-Difficile [12]. Par conséquent, plusieurs relaxations furent proposées ces deux dernières décennies afin de supprimer les valeurs de départ et de fin incohérentes en des temps polynomiaux. Parmi ces relaxations, la règle de filtrage Time-Tabling fut le sujet de nombreuses recherches par la communauté d'ordonnancement [4, 11, 14]. Un algorithme idempotent fut proposé par Letort [11] qui implémente Time-Tabling avec une complexité temporelle de $\mathcal{O}(n^2)$ et a été appliqué avec succès à des problèmes de plusieurs centaines de milliers de tâches. L'algorithme le plus rapide (non-idempotent) connu pour Time-Tabling fut proposé dans [14] et a une une complexité temporelle de $\mathcal{O}(n \log n)$. Malgré son avantage lors du passage à l'échelle, Time-Tabling souffre d'un pouvoir de filtrage limité. À l'autre extrême, les raisonnements de type Raisonnement Énergétique (ER) [3, 5, 6, 15] atteignent des filtrages forts, mais à un coût prohibitif d'une complexité temporelle de $\mathcal{O}(n^3)$. Entre ces deux extrêmes, plusieurs compromis furent proposés afin de trouver un juste équilibre entre pouvoir de filtrage et vitesse d'exécution, e.g., Edge-Finding [17, 8], Time-Table Edge-Finding [18], Time-Table Extended Edge-Finding [14] ou Not-First-Not-Last [16]. Les règles de filtrages ci-dessus sont dominées par le filtrage obtenu par le Raisonnement Énergétique, à l'exception du filtrage obtenu par Not-First-Not-Last, qui n'est par comparable à celui du Raisonnement Énergétique.

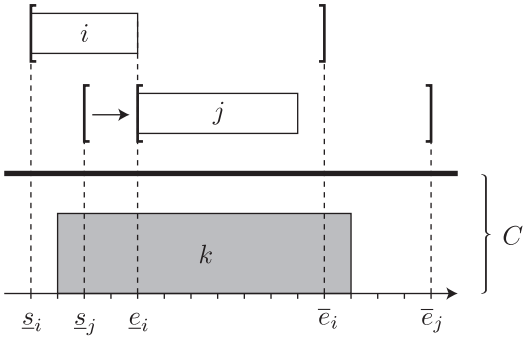

FIGURE 3 – Les tâches i et j ne peuvent pas se superposer à cause de la quantité de ressource déjà consommée par k. Comme j ne peut être assignée avant i, j doit être assignée après i : $e_i < s_j$. Cette situation ne serait pas détectée par l'approche proposée dans [2].

Étonnamment, le filtrage de type Disjunctive Reasoning (DR) [3] n'a été que partiellement adapté au contexte des ressources cumulatives. Dans [2] Baptiste et Le Pape proposent de détecter des ensembles de tâches qui ne peuvent être exécutées en parallèle sans excéder la quantité de ressource disponible initialement. Cependant, cette approche est limitée puisqu'elle ne prend pas en compte les changements dans l'utilisation de la ressource. Cette situation est illustrée dans la FIGURE 3, où une tâche k a été fixée (par la recherche ou par la propagation). Il est facile de voir que les tâches i et j ne peuvent se superposer à cause de la quantité de ressource consommée par la tâche k. Malheureusement, cette situation n'est pas détectée par l'approche proposée par Baptiste et Le Pape.

Dans le travail qui suit, nous proposons d'améliorer Disjunctive Reasoning en considérant les changements de quantité de ressource disponible au cours du temps. De même que dans les travaux de Vilim [18], nous exploitons le profil obligatoire de la ressource – un composant clef de Time-Tabling – pour détecter les paires d'activités disjointes dynamiquement. Notre nouvelle règle de filtrage – que nous appelons Time-

Table Disjunctive Reasoning – n'est dominée par aucune règle de filtrage publiée. Nous proposons un algorithme simple, qui ne repose sur aucune structure de données complexe, permettant d'implémenter cette nouvelle règle de filtrage avec une complexité temporelle de $\mathcal{O}(n^2)$. Nous proposons également plusieurs améliorations à apporter à cet algorithme. Nos résultats, évalués sur les instances de PSPLIB [9], mettent en avant que Time-Tabling Disjunctive Reasoning est une règle de filtrage prometteuse pour les contraintes cumulative à l'état-de-l'art. En effet, utiliser notre algorithme améliore de façon conséquente la résolution de certaines instances tout en ayant un surcoût négligeable sur le temps de résolution des autres instances. Cet article est structuré de la façon suivante. La Section 2 décrit Time-Tabling et les prérequis nécessaires à la lecture de ces travaux. La Section 3 est dédiée à la règle de filtrage Time-Table Disjunctive Reasoning et présente les fondements de l'algorithme. Les résultats sont présentés en Section 4. Enfin, cet article conclut sur les travaux à venir et sur plusieurs améliorations possibles.

2 Profil obligatoire

Même les tâches qui ne sont pas encore fixées apportent des informations qui peuvent être utilisées par les règles de filtrages. Par example, il est possible de déterminer un intervalle sur lequel les tâches disposant d'une fenêtre d'exécution étroite consommeront toujours de la ressource. Un tel intervalle est appelé *partie obligatoire*.

Définition 1 (Partie Obligatoire). *Considérons une tâche $i \in \mathcal{T}$. La partie obligatoire de i est l'intervalle de temps $[\overline{s}_i, \underline{e}_i[$. En d'autres mots, la tâche i possède une partie obligatoire si et seulement si son temps de départ maximum est inférieur à son temps de fin minimum.*

Si la tâche i a une partie obligatoire, nous savons que la tâche i va consommer c_i de ressource pendant toute sa partie obligatoire peu importe le moment auquel la tâche sera réellement exécutée (voir FIGURE 4).

Une version minimale de la consommation de la ressource sur le temps peut être facilement obtenue en agrégeant les partie obligatoires de toutes les tâches. [1] Cette agrégation est généralement appelée profil obligatoire de la ressource.

Définition 2 (Profil obligatoire). *Le profil obligatoire d'une ressource $TT_{\mathcal{T}}$ correspond à l'agrégation des parties obligatoires de toutes les tâches contenues dans $\mathcal{T}$.*

1. Notons que la partie obligatoire d'une tâche fixée correspond à cette même tâche.

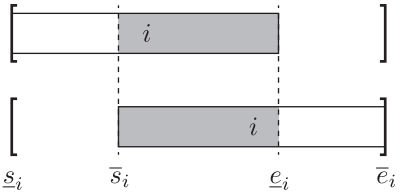

FIGURE 4 – La tâche i a une partie obligatoire $[\overline{s}_i; \underline{e}_i[$ si sa date de début maximale est strictement plus petite que sa date de fin minimale : $\overline{s}_i < \underline{e}_i$. La tâche i consomme nécessairement de la ressource pendant sa partie obligatoire, quel que soit son ordonnancement final.

Cette notion peut être définie formellement comme une fonction escalier :

$$TT_{\mathcal{T}} = t \in \mathbb{Z} \rightarrow \sum_{i \in \mathcal{T} | \overline{s}_i \leq t < \underline{e}_i} c_i. \qquad (2)$$

Le problème est incohérent si $\exists t \in \mathbb{Z} : TT_{\mathcal{T}}(t) > C$.

Le profil obligatoire peut être calculé en $\mathcal{O}(n)$ au moyen d'un algorithme de balayage pour peu que l'on dispose des tâches triées par temps de départ maximum et temps de fin minimum.

3 Time-Table Disjunctive Reasoning

Afin d'expliquer Time-Table Disjunctive Reasoning, nous comptons utiliser quelques notations additionnelles. Pour ce faire, considérons deux intervalles de temps I et J. Nous utilisons le terme J contient I si la relation $I \subseteq J$ est vérifiée. Par ailleurs, nous disons que I et J se superposent si et seulement si $I \cap J \neq \emptyset$.

3.1 Raisonnement disjonctif et intervalle minimal de superposition

Dans [3], Baptiste et al. décrivent brièvement le raisonnement disjonctif dans le contexte cumulatif comme un raisonnement entre toutes les paires de tâches $i \neq j$, qui force la cohérence-de-bornes sur la formule suivante :

$$c_i + c_j \leq C \quad \vee \quad e_i \leq s_j \quad \vee \quad e_j \leq s_i \qquad (3)$$

La règle de filtrage pour mettre à jour le temps de départ d'un tâche peut être trivialement dérivée de cette formule de prédicats.

Proposition 1 (Raisonnement disjonctif). *Considérons une paire de tâches $i \neq j \in \mathcal{T}$ telle que $c_i + c_j > C$, $\underline{s}_j < \underline{e}_i$ et $\overline{s}_i < \underline{e}_j$. Alors, le prédicat $e_i \leq s_j$ doit être vérifié.*

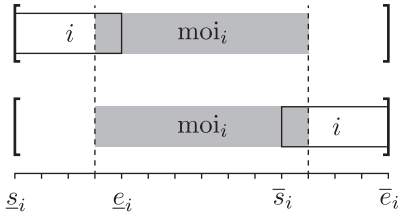

FIGURE 5 – Intervalle minimal de superposition de la tâche i. Quel que soit l'endroit où i est placé, i doit se superposer à moi_i, et moi_i est le plus petit intervalle qui a cette propriété.

Cette règle affirme que si les tâches i et j ne peuvent se superposer et que si j ne peut être ordonnancée au plus tôt sans se superposer à i, alors j ne peut commencer avant la fin minimale de i : $\underline{e_i} \leq \underline{s_j}$. Dans ce contexte, la tâche i est nommée *tâche poussante* alors que la tâche j est appelée *tâche poussée*. Une règle de filtrage pour ajuster la date de fin maximale peut trivialement être dérivée de cette première règle par symétrie.

La règle présentée en PROPOSITION 1 effectue une partie du travail de Time-Tabling. En effet, lorsque $c_i + c_j > C$ et que la tâche i a une partie obligatoire, le raisonnement sur la paire de tâches (i, j) est dominé par celui de Time-Tabling [3]. Cependant, un filtrage supplémentaire peut être dérivé par raisonnement disjonctif lorsque la tâche i n'a pas de partie obligatoire. Cette déduction peut être faite lorsque placer j à son temps de départ minimum causerait une superposition avec i peu importe la position de i. L'idée fondamentale revient à dire que j ne peut s'exécuter sur un intervalle de temps sur lequel i s'exécute au moins à un moment donné. Une contrepartie à la partie obligatoire de i peut donc être définie. Nous la nommons intervalle minimal de superposition de i.

Définition 3. *L'intervalle minimal de superposition d'une tâche i, dénoté moi_i, est l'intervalle de temps le plus petit tel que i s'exécute au moins durant un point de temps de cet intervalle, et ce peu importe le moment auquel la tâche i est exécutée. Formellement, l'intervalle minimal de superposition est défini par $[\underline{e_i} - 1, \overline{s}_i]$.*

L'intervalle minimal de superposition d'une tâche i, moi_i, est illustré en FIGURE 5.

Lorsqu'une tâche possède une partie obligatoire, nous considérons qu'elle ne dispose pas d'intervalle minimal de superposition. En utilisant ce nouveau concept, il est possible de reformuler la partie de la PROPOSITION 1 spécifique au raisonnement disjonctif.

Proposition 2 (Raisonnement disjonctif restreint)**.** *Considérons une paire de tâches $i \neq j$ telle que la tâche i n'a pas de partie obligatoire ($\underline{e_i} \leq \overline{s}_i$) et que $c_i + c_j > C$. Si ordonnancer la tâche j à son temps de départ minimum la fait se superposer complètement à l'intervalle minimal de superposition de i ($\text{moi}_i \subseteq [\underline{s_j}, \underline{e_j}[$), alors le prédicat $\underline{e_i} \leq s_j$ doit être vérifié.*

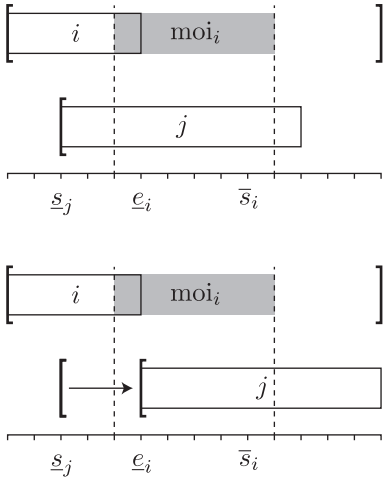

FIGURE 6 – En haut, les tâches i et j ne peuvent pas se superposer pour des raisons de capacité. Assigner s_j à $\underline{s}_j$ forcerait j à contenir moi_i, ce qui rendrait le placement de i impossible. En bas, les valeurs incohérentes de s_j sont filtrées, ce sont les $t \leq \min(\text{moi}_i)$. Ce filtrage est fait en changeant la valeur de $\underline{s}_j$ à $\underline{e}_i$

.

Cette règle présentée en PROPOSITION 2 est illustrée par la FIGURE 6. L'algorithme suivant est directement dérivé de cette règle.

Algorithm 1: Un algorithme appliquant la règle présentée en PROPOSITION 2 en $\mathcal{O}(n^2)$.

Input: un ensemble de tâches $\mathcal{T}$, capacity C
Input: la capacité de la ressource C
Input: un tableau s liant chaque tâche i à $\underline{s}_i$
Output: un tableau s' liant chaque tâche i à son nouveau temps de départ.

```
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)
```

3.2 Time-Table Disjunctive Reasoning restreint

L'une des faiblesses du raisonnement disjonctif réside dans son incapacité à prendre en compte les changements dans la quantité de ressource disponible avec

le temps (voir FIGURE 3). Dans cette section, nous montrons comment exploiter l'information contenue dans le profil obligatoire (voir Section 2) pour proposer une version renforcée du raisonnement disjonctif que nous appelons *Time-Table Disjunctive Reasoning*.[2]

Nous commençons par introduire Time-Table Disjunctive Reasoning pour un cas simple où les paires de tâches considérées ne possèdent aucune partie obligatoire et ne contribuent donc pas au profil obligatoire. Ce cas particulier nous évite de devoir supprimer les contributions de chaque tâche du profil lorsque nous appliquons le raisonnement disjonctif. Cette restriction sera levée dans la Section 3.3 afin de considérer n'importe quelle paire de tâches.

Considérons donc une paire de tâches $i \neq j$ telle que ni i ni j ne possède de partie obligatoire et que $\text{moi}_i \subseteq [\underline{s}_j, \underline{e}_j[$. Alors, PROPOSITION 2 ne fait que comparer $c_i + c_j$ à C. Cependant, les tâches contenues dans $\mathcal{T} - \{i, j\}$ pourraient ne pas laisser C unités de ressources disponibles au moment où i et j se superposent. Nous pouvons en dériver une nouvelle règle qui prend les parties obligatoires des tâches contenues dans $\mathcal{T} - \{i, j\}$ en compte.

Proposition 3. *Considérons une paire de tâches $i \neq j \in \mathcal{T}$ telle que ni i ni j ne possède de partie obligatoire et que $c_i + c_j + \min_{t \in \text{moi}_i} TT_{\mathcal{T}}(t) > C$. Si exécuter la tâche j à son temps de départ minimum la fait contenir l'intervalle minimal de superposition de i ($\text{moi}_i \subseteq [\underline{s}_j, \underline{e}_j[$), alors le prédicat $\underline{e}_i \leq \overline{s}_j$ doit être vérifié.*

Démonstration. Si la tâche j contient moi_i, alors j devrait augmenter la consommation du profil obligatoire de c_j unité pendant tout moi_i (la tâche j ne possède pas de partie obligatoire et ne contribue donc pas au profil obligatoire). Alors, placer la tâche i n'importe où devrait également augmenter la consommation de la ressource de c_i unités pendant au moins un point de temps $t \in \text{moi}_i$, rendant ainsi la consommation totale au point t supérieure à la capacité de la ressource C. Par ailleurs, puisque $\text{moi}_i \subseteq [\underline{s}_j, \underline{e}_j[$, la durée de j est telle que la placer avant $\underline{e}_i$ la ferait contenir moi_i. Par conséquent, ces valeurs sont incohérentes et le prédicat $\underline{e}_i \leq \overline{s}_j$ doit être vérifié. □

3.3 Time-table Disjunctive Reasoning

La règle de filtrage présentée en PROPOSITION 2 peut être renforcée en utilisant une idée similaire à celle proposée dans [14, 18]. Pour ce faire, nous divisons chaque tâche en deux parties distinctes : sa partie obligatoire et sa *partie libre*.

2. L'idée d'exploiter le profil obligatoire pour renforcer des règles de filtrage existante a déjà été suivie avec succès dans [14, 18].

Définition 4 (Partie libre). *Considérons une tâche $i \in \mathcal{T}$ telle que i possède une partie obligatoire ($\overline{s}_i < \underline{e}_i$). Sa partie libre, dénotée i_f, correspond à une tâche séparée dont le temps de départ minimum et le temps de fin maximum sont les même que ceux de i : $\underline{s}_{i_f} = \underline{s}_i$ and $\overline{e}_{i_f} = \overline{e}_i$. La durée de i_f est égale à la durée de i moins la taille de sa partie obligatoire : $d_{i_f} = d_i - (\underline{e}_i - \overline{s}_i)$. Clairement, si i n'a pas de partie obligatoire alors $i = i_f$.*

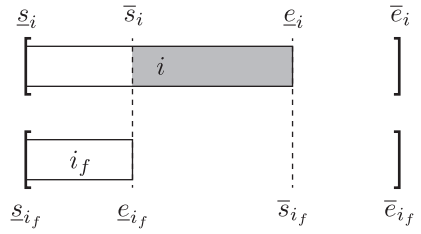

FIGURE 7 – Une tâche i avec sa partie obligatoire et sa partie libre i_f. La partie libre i_f a toujours un intervalle minimal de superposition moi_{i_f}.

Par définition, la partie libre d'une activité n'a pas de partie obligatoire et possède donc toujours un intervalle minimal de superposition (voir FIGURE 7). Par la suite, nous référençons l'ensemble des tâches possédant une partie libre strictement positive par $\mathcal{T}_f = \{i_f \mid i \in \mathcal{T} \wedge d_{i_f} > 0\}$.

Les parties libres nous permettent de généraliser PROPOSITION 2 à toutes les tâches sans devoir se soucier de la présence de parties obligatoires.

Proposition 4 (Time-Table Disjunctive Reasoning). *Considérons une paire de tâches $i_f \neq j_f \in \mathcal{T}_f$ telle que $c_i + c_j + \min_{t \in \text{moi}_{i_f}} TT_{\mathcal{T}}(t) > C$. Si exécuter la tâche j_f à son temps de départ minimum la fait se superposer à l'intervalle minimal de superposition de i_f ($\text{moi}_{i_f} \subseteq [\underline{s}_{j_f}, \underline{e}_{j_f}[$), alors le prédicat $\underline{e}_{i_f} \leq \underline{s}_j$ doit être vérifié.*

Démonstration. Supposons que la partie gauche de l'implication soit vérifiée pour i et j et supposons également que $s_j \leq \min(\text{moi}_i)$. Puisque j_f contient moi_{i_f} lorsque celle-ci est placée à son temps de départ minimum et $d_j \geq d_{j_f}$, placer j avant ou à $\min(\text{moi}_{i_f})$ la fait contenir moi_{i_f}. Observons que j n'a pas de partie obligatoire pendant moi_{i_f}, puisque $\max(\text{moi}_{i_f}) \leq \underline{e}_{j_f} = \min(\underline{e}_j, \overline{s}_j)$.

Si i ne possède pas de partie obligatoire, i ne contribue pas au profil obligatoire. Cela signifie que $\forall t \in \text{moi}_{i_f}, TT(t) = TT_{\mathcal{T}-\{i,j\}}(t)$. Dès lors, placer j avant ou à $\min(\text{moi}_i)$ augmente l'utilisation de la ressource durant moi_{i_f} de c_j unités, ce qui contredit le fait que $i = i_f$ soit placée sur son intervalle minimal de superposition.

Si i possède une partie obligatoire, cette tâche contribue au profil obligatoire exactement durant l'intervalle $[\min(\text{moi}_{i_f}) + 1, \max(\text{moi}_{i_f}) - 1]$. Dès lors, placer j avant ou à $\min(\text{moi}_i)$ augmente l'utilisation de la ressource aux points de temps $\min(\text{moi}_{i_f})$ et $\max(\text{moi}_{i_f})$ de c_j unités. Cela empêche i_f d'être placée vers la gauche ou vers la droite, ce qui par conséquent signifie que la tâche i elle même ne peut pas s'exécuter durant ces points de temps. Cela contredit la définition d'intervalle minimal de superposition puisque i s'exécute pendant au moins un point de temps de cet intervalle. □

L'utilisation des parties libres a deux avantages : (i) Elle permet d'utiliser n'importe quelle tâche i en tant que *tâche poussante*, (ii) Elle permet de ne calculer qu'un fois la hauteur minimale par tâche poussante sans devoir supprimer les parties obligatoires relatives à la contribution de j.

Nous proposons donc un second algorithme capable de tirer avantage des parties obligatoires et libres pour appliquer la règle de filtrage de la PROPOSITION 4 avec une complexité temporelle de $\mathcal{O}(n^2)$. Dans cet algorithme, le profil obligatoire (TT) est encodé dans un simple tableau de paires (t, c) ou t correspond à une date et c à une consommation. Ce tableau est ordonné de façon strictement croissante selon les dates. La primitive $\mathit{consumption}(i, TT)$ (voir ligne 3) peut-être implémentée via la formule $\min_{t \in \text{moi}_{i_f}} TT(t)$, calculable de façon linéaire dans la taille du profil obligatoire. Dès lors, la complexité de cet algorithme est de $\mathcal{O}(n^2)$.

Algorithm 2: Time-Table Disjunctive filtering algorithm.

Input: l'ensemble des tâches $\mathcal{T}$
Input: l'ensemble des tâches poussantes $\mathit{pushing} \subseteq \mathcal{T}_f$
Input: l'ensemble des tâches poussées $\mathit{pushed} \subseteq \mathcal{T}_f$
Input: la capacité de la ressource C
Input: un tableau s' liant chaque tâche i à $\underline{s}_i$
Output: un tableau s' liant chaque tâche i à son nouveau temps de départ.

```
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))
```

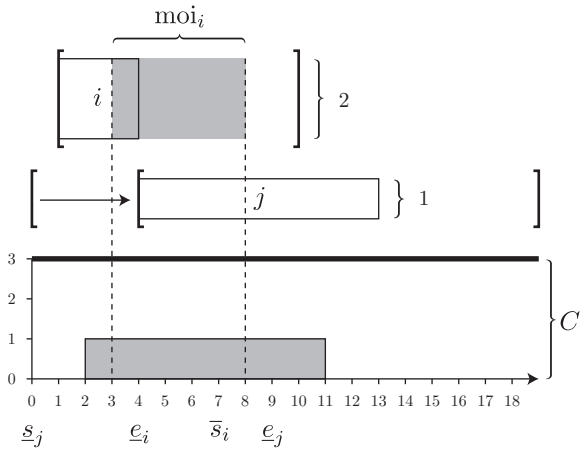

FIGURE 8 – À cause du profil obligatoire, les tâches i et j ne peuvent se superposer. Comme j ne peut être placée avant i, j doit être assignée après i : $e_i \leq s_j$.

Proposition 5. *Le filtrage de TTDR n'est dominé ni par le Raisonnement Énergétique, ni par Not-First-Not-Last.*

Démonstration. FIGURE 8 présente un exemple où TTDR peut filtrer certaines valeurs incohérentes que ni le Raisonnement Énergétique ni Not-First-Not-Last ne sont capables de détecter. Soit deux tâches i et j définie respectivement par $(\underline{s}_i, \overline{e}_i, d_i, c_i) = (2, 11, 3, 2)$, et $(\underline{s}_j, \overline{e}_j, d_j, c_j) = (1, 20, 9, 1)$. Ni la tâche i ni la tâche j ne possède de partie obligatoire i.e. $i_f = i$ et $j_f = j$. Observons également que la condition $\text{moi}_i = [4, 8] \subseteq [\underline{s}_j, \underline{e}_j[= [1, 10[$ est respectée et que le minimum de TT durant $\text{moi}_i = [4, 8]$ est de 1. Cela signifie que les tâches i et j, consommant respectivement 2 et 1 unités de ressource, ne sont pas autorisées à se superposer en $[4, 8]$. Dès lors, $\underline{s}_j$ est mis à jour de sorte que $\min(\text{moi}_i) + 1 = \underline{e}_i = 5$. □

4 Résultats et expériences

Les performances de Time-Table Disjunctive Reasoning (TTDR) furent évaluées sur une célèbre panoplie de tests du Resource Constraints Project Scheduling Problem (RCPSP) de PSPLIB [9]. Le but de ces expériences était de mesurer les performances de plusieurs combinaisons de règles de filtrage de la contrainte cumulative avec et sans TTDR. Voici la liste exhaustive des règles de filtrage considérées :

— Time-Tabling (TT), implémenté au moyen d'une variante rapide de l'algorithme présenté dans [11] ;
— Time-Table Disjunctive Reasoning (TTDR), implémenté selon l'algorithme présenté dans ce papier et les différentes améliorations proposées

dans [7] ;
— Edge-Finding (EF), implémenté comme proposé dans [8] ;
— Raisonnement Énergétique (ER), implémenté avec un algorithme cubique proposé par Baptiste et al. in [3]. Nous avons ajouté quelques améliorations proposées par Derrien et Petit pour réduire le nombre d'intervalles considérés [5] ;
— Not-First-Not-Last (NFNL), implémenté avec un algorithme $\mathcal{O}(n^3)$, une variante de celui proposé par Schutt et al. [16].

Toutes ces règles de filtrage furent implémentées et testées dans le solver open-source OscaR [13]. Nous avons utilisé une recherche SetTimes classique [10], en brisant les égalités par le choix d'une tâche de durée minimale. Ces résultats ont été calculés au moyen d'un processeur 4-core, 8 thread Core(TM) i7-2600 CPU @ 3.40GHz et de 8GB de mémoire RAM, exécuté dans la machine virtuelle Java SE 1.7 JVM.

Nous reportons la distribution cumulée $F(\tau)$ des instances résolues dans les limites calculatoires de τ dans les figures FIGURE 9 et FIGURE 10. Dans la colonne de gauche, τ est le temps, dans celle de droite c'est le nombre de noeuds de l'arbre de recherche ; l'abscisse est logarithmique dans les deux cas $F(\tau) = k$ signifie que k instances ont été résolues en moins de τ ms ou noeuds. Nous avons fixé une limite de 90s pour chaque calcul. [3] A cause du manque de place, nous ne montrons que les résultats obtenus sur les instances à 30 et 120 tâches. Les résultats obtenus avec 60 tâches sont similaires, mais nous avons observé qu'ajouter TTDR avait peu d'effet sur les instances de 90 tâches, où seulement deux instances supplémentaires sont fermées en ajoutant TTDR.

Malgré sa complexité temporelle de $\mathcal{O}(n^2)$, l'algorithme proposé pour TTDR reste très léger. L'unique surcoût de calcul apparait sur de très petits valeurs de temps, lorsque TTDR n'est utilisé qu'en combinaison avec TT. Cependant, le filtrage supplémentaire apporté par TTDR compense rapidement ce léger surcoût et permet de résoudre plus d'une instance pour une limite de temps τ.

Les instances PSPLIB sont connues pour être plus disjonctives que cumulatives. [4] Ajouter des raisonnements à base d'énergie aux instances PSPLIB est un compromis risqué. En effet, les raisonnements à base d'énergie trivialisent peu les instances PSPLIB, alors que TTDR améliore le nombre d'instances résolues par TT seul. Ceci confirme expérimentalement que TTDR

3. C'est pourquoi les graphes de temps n'ont plus de points après 2^{17}ms.

4. Un problème est dit hautement disjonctif lorsque beaucoup de paires de tâches ne peuvent être exécutées en parallèle ; au contraire, un problème est hautement cumulatif si beaucoup de paires de tâches peuvent être exécutées en parallèle [2].

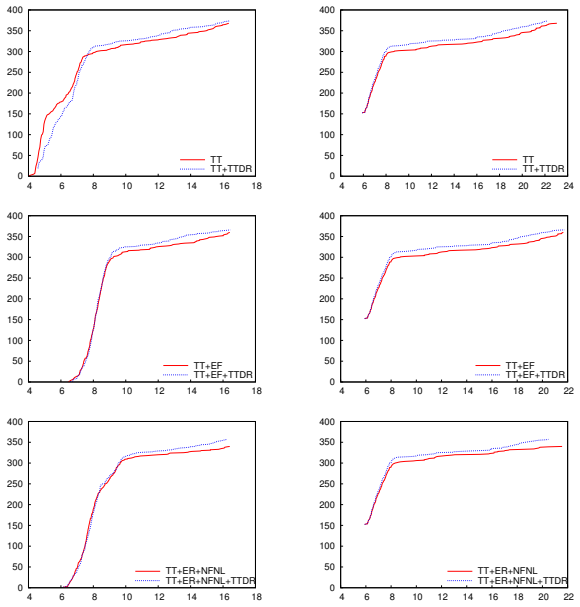

FIGURE 9 – Graphiques pour les instances de PSPLIB30. En ordonnée, le nombre cumulé d'instances résolues. Dans la colonne de gauche, l'abscisse est $log_2(t)$, avec t le temps en ms. Dans la colonne de droite, l'abscisse est $log_2(n)$, avec n le nombre de noeuds pour trouver l'optimal et prouver l'optimalité.

est complémentaire aux filtrages à base d'énergie pour la contrainte `cumulative` (voir Prop. 5)

Finalement, nous voyons aussi que le gain en termes d'instances résolues ne baisse pas quand nous passons des instances à 30 tâches à celles à 120 tâches. Ceci signifie que le filtrage dépend plus de la nature des instances que de leur taille, et que l'algorithme $\mathcal{O}(n^2)$ pour TTDR passe bien à l'échelle.

5 Conclusion

Cet article introduit le Time-Table Disjunctive Reasoning, une nouvelle règle de filtrage qui utilise le profil obligatoire pour détecter des paires disjonctives dynamiquement. En s'appuyant sur les intervalles minimaux de superposition, ce raisonnement accomplit un nouveau type de filtrage qui n'est dominé par aucune règle de filtrage existante telle que le Raisonnement Énergétique ou Not-First-Not-Last. En plus d'être nouvelle, TTDR peut être implémentée avec un simple algorithme $\mathcal{O}(n^2)$ qui ne repose sur aucune structure de données complexe. Les bénéfices de l'utilisation de TTDR en complément à d'autres règles de filtrage ont été évalués sur les instances classiques de PSPLIB. Nos résultats montrent que TTDR est une règle de filtrage prometteuse pour étendre les implémenta-

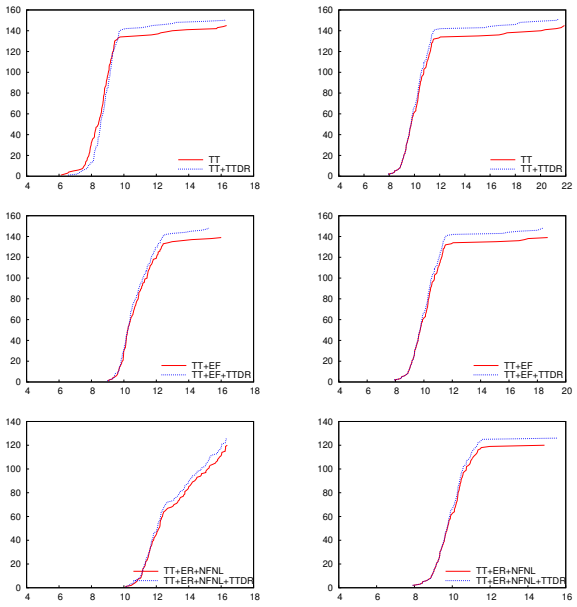

FIGURE 10 – Graphiques pour les instances de PSPLIB120. En ordonnée, le nombre cumulé d'instances résolues. Dans la colonne de gauche, l'abscisse est $log_2(t)$, avec t le temps en ms. Dans la colonne de droite, l'abscisse est $log_2(n)$, avec n le nombre de noeuds pour trouver l'optimal et prouver l'optimalité.

tions état de l'art de la contrainte `cumulative`. En effet, utiliser notre algorithme peut améliorer le processus de résolution de certaines instances substantiellement, et n'ajoute qu'un surcoût dérisoire aux autres.

Bien que le renforcement proposé par TTDR soit déjà un bon compromis en termes de vitesse/filtrage, des améliorations peuvent être envisagées. Par exemple, sa complexité en pratique pourrait être réduite en utilisant des techniques de balayage pour éviter l'examen de paires qui ne se superposent pas. Une amélioration encore plus intéressante concernerait le filtrage : on pourrait le rendre encore plus fort en considérant des intervalles minimaux de superposition de plus d'une tâche à la fois.

Remerciements. Steven Gay est financé par le projet Innoviris 13-R-50 de la région Bruxelles-Capitale. Renaud Hartert est un aspirant du FRS-FNRS. Les auteurs voudraient remercier Sascha Van Cauwelaert pour avoir partagé ses instances et son parseur.

Références

[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] Jacques Erschler, Pierre Lopez, and Catherine Thuriot. Scheduling under time and resource constraints. In *Proc. of Workshop on Manufacturing Scheduling, 11th IJCAI, Detroit, USA*, 1989.

[7] Steven Gay, Renaud Hartert, and Pierre Schaus. Time-table disjunctive reasoning for the cumulative constraint. In *CPAIOR*. Springer, 2015.

[8] 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.

[9] Rainer Kolisch, Christoph Schwindt, and Arno Sprecher. Benchmark instances for project scheduling problems. In *Project Scheduling*, pages 197–212. Springer, 1999.

[10] Claude Le Pape, Philippe Couronné, Didier Vergamini, and Vincent Gosselin. Time-versus-capacity compromises in project scheduling, 1994.

[11] 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.

[12] Wilhelmus Petronella Maria Nuijten. *Time and resource constrained scheduling : a constraint satisfaction approach*. PhD thesis, Technische Universiteit Eindhoven, 1994.

[13] OscaR Team. OscaR : Scala in OR, 2012. Available from `bitbucket.org/oscarlib/oscar`.

[14] 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.

[15] Eric Pinson. *Le problème de job-shop*. PhD thesis, Paris 6, 1988.

[16] 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.

[17] 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.

[18] 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.