Algorithmes de filtrage pour des problèmes cumulatifs discrets avec dépassements de ressource

Alexis De Clercq Thierry Petit Nicolas Beldiceanu Narendra Jussien

Mines-Nantes, LINA UMR CNRS 6241,
4, rue Alfred Kastler, FR-44307 Nantes, France.

{Alexis.De-Clercq, Thierry.Petit, Nicolas.Beldiceanu, Narendra.Jussien}@mines-nantes.fr

Abstract

Dans de nombreux problèmes cumulatifs, l'horizon est fixé et ne peut être retardé. Dans cette situation, il arrive souvent que toutes les activités ne puissent pas être ordonnancées sans dépasser la capacité en certains points de temps. De plus, cette capacité n'est pas nécessairement la même sur toute la fenêtre temporelle. Dans ce contexte, cet article introduit une nouvelle contrainte pour résoudre cette classe de problèmes. Nous adaptons deux algorithmes à notre contexte : Sweep et l'algorithme d'Edge-Finding de P. Vilìm. Comme dans certains problèmes des dépassements minimaux sont imposés, nous décrivons une procédure spécifique à ce type d'événements. Nous introduisons une heuristique de recherche spécifique à notre contrainte. Nous expérimentons avec succès notre contrainte.

1 Introduction

Les problèmes d'ordonnancement consistent à ordonner des activités. En *ordonnancement cumulatif*, chaque activité a une durée et nécessite, pour son exécution, la disponibilité d'une certaine quantité de ressource renouvelable, sa *consommation* (ou *demande en capacité*). Habituellement, l'objectif est de minimiser *l'horizon* (la date de fin maximale d'une activité), tandis qu'en chaque point de temps la consommation cumulée des activités ne doit pas excéder une limite sur la ressource disponible, la *capacité*. Cependant, de nombreux problèmes industriels nécessitent que les activités soient ordonnancées dans une fenêtre temporelle donnée : l'horizon est fixé et ne peut être retardé. Dans cette situation, il peut arriver que toutes les activités ne puissent pas être ordonnancées sans dépasser la capacité en certains points de temps. Pour obtenir une solution, les dépassements peuvent être tolérés dans une certaine limite, à condition que des règles opérationnelles garantissant la faisabilité pratique de la solution soient satisfaites. De plus, dans certains problèmes, la fenêtre temporelle est partitionnée : la capacité n'est pas la même pour chaque intervalle de la partition. La Figure 1 illustre une telle situation avec des dépassements.

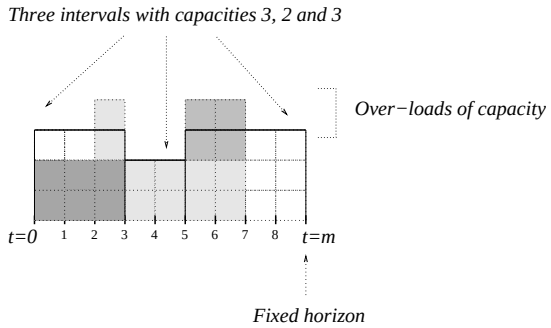

FIGURE 1 – Un problème cumulatif avec un horizon fixé ($m = 9$) et 3 intervalles avec des capacités égales respectivement à 3, 2 et 3. Chaque activité requiert 2 unités de ressource. La première commence en $t = 0$ et finit en $t = 3$, la deuxième commence en $t = 2$ et finit en $t = 7$, la troisième commence en $t = 5$ et finit en $t = 7$.Il y a deux dépassements de capacité : un dans le premier intervalle au temps 2, et un dans le troisième intervalle aux temps 5 et 6.

Nous introduisons une nouvelle contrainte, SOFT-CUMULATIVE, qui étend le travail présenté dans [9] pour considérer des problèmes cumulatifs avec dépassements dans lesquels des capacités locales sont définies pour des intervalles de temps disjoints. Plusieurs mesures de violations peuvent être utilisées pour quantifier les dépassements de capacités locales et pour calculer une valeur d'objectif : on peut vouloir minimiser le dépassement le plus haut, ou minimiser la somme des aires en dépassement. Nous décrivons des problèmes concrets qui peuvent être modélisés en uti-

lisant SoftCumulative et d'autres contraintes additionnelles. Certaines de ces contraintes peuvent mener à des situations où les dépassements sont imposés dans certains intervalles de temps.

Notre contribution principale est un nouvel algorithme de filtrage associé à SoftCumulative, qui considère aussi les dépassements imposés. Il est décomposé en trois phases. La première phase est une adaptation de l'algorithme de Sweep en $O(n \cdot log(n))$, pour Cumulative [4], où n est le nombre d'activités. Dans notre cas, la complexité temporelle dépend aussi du nombre p de capacités locales définies par l'utilisateur : $O((n+p) \cdot log(n+p))$. Les avantages du sweep sont préservés : le profil est calculé et les variables de début des activités sont filtrées en un balayage, et la complexité ne dépend pas du nombre de points de temps. Pour réaliser un raisonnement énergétique[1] complémentaire au raisonnement du Sweep, basé sur le profil, dans une deuxième phase, nous adaptons l'algorithme d'Edge-Finding de P. Vilím's [13], en $O(k \cdot n \cdot log(n))$, où k est le nombre de demandes en capacité distinctes. Dans notre cas, la complexité temporelle est $O(p \cdot k \cdot n \cdot log(n))$. Dans les deux phases les bornes inférieures de l'objectif sont incluses dans les conditions de filtrage sans augmenter la complexité. La troisième phase est une propagation spécifique aux événements survenant sur les valeurs minimales de variables exprimant des violations de capacités locales, sans augmenter la complexité temporelle.

La Section 2 présente les pré-requis utiles pour comprendre nos contributions et définit SoftCumulative. La Section 3 présente des exemples motivant l'intérêt de notre travail. La Section 4 décrit l'algorithme de filtrage de SoftCumulative. La Section 5 présente une nouvelle stratégie de recherche dédiée, ainsi que les expérimentations effectuées en utilisant choco [1].

2 Définitions et Notations

Nous considérons un ensemble A d'activités non-préemptives, c'est à dire, des activités qui ne peuvent pas être interrompues. Une activité est définie par des variables. Etant donnée une variable x, $\underline{x}$ (resp. $\overline{x}$) représente la valeur minimum (resp. maximum) dans son domaine $D(x)$.

Définition 1 (Activité) *Une activité $a_i \in A$ est définie par quatre variables : sa_i représente son début ; da_i, sa durée ; ca_i, sa date de fin telle que $ca_i = sa_i + da_i$; et $ra_i \geq 0$, la quantité discrète de ressource consommée par a_i en tout point de temps entre son début et sa fin.*

1. Déductions basées sur la comparaison entre la ressource consommée et la ressource disponible.

Nous notons $\underline{e}(a_i)$ *l'énergie minimum* d'une activité $a_i \in A$, égale à $\underline{da_i} * \underline{ra_i}$. Etant donné $\Omega \subseteq A$, $\underline{e}(\Omega)$ est l'énergie minimum de Ω, égale à $\sum_{a_i \in \Omega} \underline{e}(a_i)$. Nous notons $\underline{mins}(\Omega) = \min_{a_i \in \Omega}(\underline{sa_i})$, $\underline{maxs}(\Omega) = \min_{a_i \in \Omega}(\overline{sa_i})$, $\overline{minc}(\Omega) = \max_{a_i \in \Omega}(\underline{ca_i})$, $\overline{maxc}(\Omega) = \max_{a_i \in \Omega}(\overline{ca_i})$. *L'horizon de temps m* est la fin maximum possible entre les activités. Nous imposons $\forall a_i \in A, ca_i \leq m$. Enfin, *une instantiation de A* est une affectation de toutes les variables définissant des activités dans A. En PPC, les problèmes cumulatifs peuvent être modélisés en utilisant la contrainte Cumulative [2]. Nous rappelons d'abord sa définition avant d'introduire SoftCumulative.

Définition 2 (Hauteur) *Etant donnée une ressource et une instantiation d'un ensemble A de n activités, à chaque point de temps t la* hauteur cumulée *h_t des activités passant par t est $h_t = \sum_{a_i \in A, sa_i \leq t < ca_i} ra_i$.*

Définition 3 (Cumulative) *Etant donnée une ressource avec une capacité limitée par capa et une instantiation d'un ensemble A de n activités, la contrainte Cumulative$(A, capa)$ est satisfaite ssi les deux contraintes suivantes sont satisfaites :*

- *C1 : Pour chaque activité $a_i \in A, sa_i + da_i = ca_i$, et*
- *C2 : A chaque point de temps t, $h_t \leq capa$.*

Nous définissons maintenant la nouvelle SoftCumulative, une contrainte représentant des problèmes cumulatifs avec un horizon fixé, des variables de coût exprimant des dépassements de capacité sur des intervalles de temps, et une variable objectif agrégeant les coûts.

Définition 4 (SoftCumulative) *Considérons une ressource avec une capacité limitée par capa et un ensemble A de n activités tel que $\overline{maxc}(A) \leq m$. Nous définissons :*

- *Une partition $P = [p_0, \ldots, p_{k-1}]$ de $[0, m[$ en p intervalles consécutifs tels que chaque p_j est défini par son début et sa fin : $p_j = [sp_j, ep_j[$.*
- *Une séquence de capacités locales $Loc = [lc_0, \ldots, lc_{k-1}]$ en correspondance une à une avec P, telle qu'une capacité locale $lc_j \leq capa$ est associée avec chaque intervalle p_j.*
- *Une séquence de variables de coût $Cost = [cost_0, \ldots, cost_{k-1}]$ en correspondance une à une avec P, telle qu'une variable $cost_j$ est associée avec l'intervalle p_j.*
- *Une variable objectif obj.*
- *Un indicateur $costC \in \{max, sum\}$ indiquant comment les variables Cost sont calculées (i.e. considérant le dépassement maximum ou la surface au-dessus).*

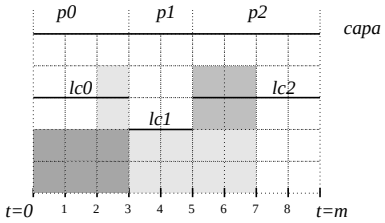

FIGURE 2 – Une instance de SOFTCUMULATIVE représentant le problème de la Figure 1. Si $costC = max$, $cost_0 = 1$, $cost_1 = 0$, $cost_2 = 1$ et si $objC = max$, $obj = 1$, si $objC = sum$, $obj = 2$. Si $costC = sum$, $cost_0 = 1$, $cost_1 = 0$, $cost_2 = 2$. Si $objC = max$, et si $objC = max$, $obj = 2$, si $objC = sum$, $obj = 3$

- *Un indicateur* $objC \in \{max, sum\}$ *indiquant comment la variable obj est calculée (i.e. considérant le maximum ou la somme des variables Cost).*

Etant donnée une instantiation de A, la contrainte SOFTCUMULATIVE$(A, capa, P, Loc, Cost, obj, costC, objC)$ *est satisfaite ssi les quatre conditions suivantes sont satisfaites :*

- *C1 et C2 (voir* Définition 3*).*
- *C3 :* $\forall j \in [0, k-1]$ *:* $cost_j = costC_{t \in p_j}(\max(0, h_t - lc_j))$
- *C4 : Une contrainte sur l'objectif :* $obj = objC_{j \in [0,k-1]}(cost_j)$.

Comparée à SOFTCUMULATIVESUM [9], notre contrainte partage l'horizon de temps en différents intervalles de longueur et capacité propres. Les dépassements sont quantifiés par une variable de coût sur chaque intervalle, selon $costC$, et une variable objectif agrégeant les coûts selon $objC$(voir la Figure 2).

Notation 1 *Pout un point de temps t donné,* $p_j(t) = [sp_j(t), ep_j(t)[$, $cost_j(t)$ *et* $lc_j(t)$ *sont, respectivement, l'intervalle* $p_j \in P$, *la variable* $cost_j \in Cost$ *et la capacité locale* $lc_j \in Loc$ *tels que* $t \in p_j$.

3 Problèmes pratiques

Les problèmes cumulatifs avec un horizon fixé peuvent impliquer des contraintes additionnelles sur les variables *Cost*, pour distinguer les solutions ayant un intérêt pratique des solutions qui seront rejetées par l'utilisateur final [9]. Nous présentons deux cas classiques.

Répartition équitable des dépassements. Dans de nombreuses applications, *e.g.*, des problèmes d'emplois du temps où les employés doivent effectuer des heures supplémentaires pour finir leurs activités dans des périodes surchargées, les dépassements de capacité doivent être équitablement répartis. Ce besoin a été mis en évidence dans [11]. Plus tard, différentes contraintes globales ont été introduites pour équilibrer des solutions [8, 10, 12]. Pour les cas simples, des contraintes de cardinalité classiques peuvent être utilisées. En utilisant SOFTCUMULATIVE, on peut définir une partition sur l'ensemble *Cost* et imposer que, dans chaque classe de la partition, le nombre minimum de variables de coût égales à 0 soit strictement positif (voir Figure 3).

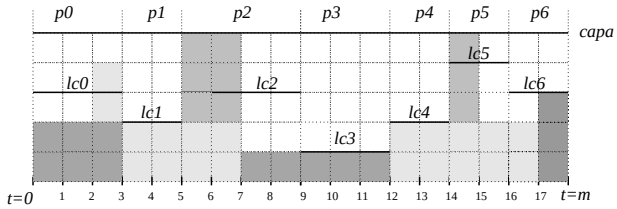

FIGURE 3 – Une solution satisfaisant des contraintes de cardinalité assitionnelles sur la partition $\{(cost_0, cost_1), (cost_2, cost_3), (cost_4, cost_5, cost_6)\}$ avec $costC = max$: au moins un coût égal à 0.

Lissage des variations de coût. Une requête fréquente consiste à limiter le nombre de variations importantes par rapport aux dépassements. Cela peut être fait en ajoutant, sur les variables *Cost*, une instance de SMOOTH$(N, tol, Cost)$ [3], où N est une variable et tol un entier. Elle impose que le nombre de fois où $|cost_{i+1} - cost_i| > tol$ soit égal à la valeur de N (i.e. N représente la limite). Par exemple, quand une entreprise embauche des intérimaires, leur nombre ne doit pas varier de manière trop importante d'un jour à l'autre. La Figure 3 est une solution satisfaisant à la fois SOFTCUMULATIVE et SMOOTH$(N, tol, Cost)$ avec $N = 1$ et $tol = 1$. Dans certaines applications, imposer des contraintes primitives de type $|cost_{i+1} - cost_i| \leq tol$ suffit.

Les dépassements mènent à des violations de capacités et les problèmes avec horizon fixés peuvent être vus comme des problèmes sur-contraints. Dans [9], les expérimentations montrent que, dans des problèmes sur-contraints avec contraintes additionnelles sur les variables représentant les violations, propager efficacement les contraintes additionnelles est indispensable pour résoudre les instances. Dans le cas de solutions équitablement réparties, une telle propagation correspond à des événements générés lorsque la valeur maximale d'une variables de coût est réduite. Plus généralement, lorsque l'on résout des problèmes sur-contraints, il est admis que les violations devraient être minimisées et ne sont jamais imposées [7]. L'exemple du lissage des variations de coût contredit cette suppo-

sition. Considérons le cas où la contrainte primitive $|cost_{i+1} - cost_i| \leq 1$ est imposée. Si $cost_i = 2$ alors il est nécessaire d'avoir $cost_{i+1} \geq 1$. En conséquence, pour être efficace dans tous les contextes possibles, le filtrage de SoftCumulative doit gérer les événements sur les valeurs minimales des variables *Cost*. La Section 4.5 présente un algorithme de filtrage dédié à de tels événements. Dans l'état de l'art [5], ce type de propagation est étudié pour SoftAllDifferent et SoftAllEqual.

4 Algorithmes de filtrage

4.1 Pré-requis

4.1.1 Sweep pour Cumulative

Le but est de réduire les domaines des variables de début en fonction du profil cumulatif, qui est construit à partir des parties obligatoires.

Définition 5 (Partie obligatoire [6]) *La* partie obligatoire $cp(a_i)$ *d'une activité* $a_i \in A$ *est l'intersection de tous les horaires possibles de* a_i*. Elle est définie par* $[\overline{sa_i}, \underline{ca_i}[$ *(de ce fait, elle est non vide ssi* $\overline{sa_i} < \underline{ca_i}$*), et une hauteur égale à* $\underline{ra_i}$ *on* $[\overline{sa_i}, \underline{ca_i}[$*, et nulle ailleurs.*

Définition 6 (Profil cumulatif) *Le* profil cumulatif *CumP est la consommation minimale cumulée, dans le temps, de toutes les activités. Pour un point de temps donné t, la hauteur de CumP en t égale à* $\sum_{a_i \in A, t \in [\overline{sa_i}, \underline{ca_i}[} \underline{ra_i}$*. C'est à dire, la somme des contributions de toutes les parties obligatoires passant par t.*

L'algorithme de sweep [4] déplace une ligne verticale Δ sur l'axe du temps, d'un événement à un autre. En un balayage, il construit le profil cumulatif et filtre les activités de manière à ne pas dépasser *capa*. Un *événement* correspond soit au début ou à la fin d'une partie obligatoire, soit à la date de début au plus tôt $\underline{sa_i}$ d'une activité $a_i \in A$. Tous les événements sont initialement calculés et triés par ordre croissant de leur date. La position de Δ est δ. A chaque pas de l'algorithme, une liste $ActToPrune$ contient les activités à filtrer.

- Les événements de partie obligatoire sont utilisés pour construire *CumP* : Tous les événements de ce type à la date δ mettent à jour la hauteur sum_h du rectangle courant dans *CumP*, en ajoutant la hauteur de l'activité si c'est le début d'une partie obligatoire ou la soustrayant si c'en est la fin. Le premier événement de partie obligatoire avec une date strictement supérieure à δ donne la fin δ' du rectangle, finalement noté $\langle [\delta, \delta'[, sum_h \rangle$.
- Les événements correspondant aux dates de début au plus tôt $\underline{sa_i}$ tels que $\delta \leq \underline{sa_i} < \delta'$ ajoutent de nouvelles activités candidates au filtrage, en fonction de $\langle [\delta, \delta'[, sum_h \rangle$ et *capa* (activités passant par $\langle [\delta, \delta'[, sum_h \rangle$). Elles sont ajoutées à la liste $ActToPrune$.

Pour chaque $a_i \in ActToPrune$ n'ayant pas de partie obligatoire dans le rectangle $\langle [\delta, \delta'[, sum_h \rangle$, si sa hauteur est plus grande que $capa - sum_h$ alors on filtre sa date de début de manière à ce que a_i ne touche pas le rectangle courant de *CumP*. Si $\overline{ca_i} \leq \delta'$ alors a_i est retirée de $ActToPrune$. Après avoir filtré les activités, δ est mise à jour par δ'.

Règle de filtrage 1 *Si* $a_i \in ActToPrune$ *n'a pas de partie obligatoire dans* $\langle [\delta, \delta'[, sum_h \rangle$ *et* $sum_h + \underline{ra_i} > capa$ *alors* $]\delta - \underline{da_i}, \delta'[$ *peut être retiré de* $D(sa_i)$*.*

La complexité temporelle de Sweep est $O(n \cdot log(n))$ où n est le nombre d'activités.

4.1.2 Edge-Finding pour Cumulative

Cette section résume l'algorithme d'edge-finding en deux phases de Vilìm [13]. Il détecte d'abord les précédences induites entre les activités, et filtre ensuite les dates de débuts d'activités. Dans les deux phases, il utilise un raisonnement énergétique : Il compare la ressource requise par un ensemble d'activités dans un intervalle donné $I = [a, b[$ de points de temps avec la ressource disponible dans cet intervalle. Cette ressource disponible est donnée par la capacité *capa* et la taille de l'intervalle.

Définition 7 (Aire disponible). *Etant donné un intervalle de temps* $I = [a, b[$*, nous notons* $Area(a, b)$ *la ressource maximale disponible, égale à* $(b - a) * capa$*.*
(Enveloppe énergétique). *Soit* Ω *un ensemble d'activités tel que* $\Omega \subseteq A$*.* L'enveloppe énergétique *de* Ω *est :* $Env(\Omega) = \max_{\Theta \subseteq \Omega}(Area(0, \underline{mins}(\Theta)) + \underline{e}(\Theta))$*.*
(Précédence). *Une activité* $a_i \in A$ finit avant la fin *d'une activité* $a_j \in A$ *ssi, dans toutes les solutions,* $ca_i \leq ca_j$*. Cette relation est notée* $a_i \lessdot a_j$*. Elle peut être étendue à un ensemble d'activités* Ω *:* $\Omega \lessdot a_j$*.*

Pour détecter les précédences par rapport à une activité a_j, nous considérons toutes les activités a_i ayant une date de fin au plus tard $\overline{ca_i}$ plus petite ou égale à $\overline{ca_j}$.

Définition 8 (Coupe à gauche) *La* coupe à gauche *de A par une activité* $a_j \in A$ *est un ensemble d'activités telles que :* $LCut(A, a_j) = \{a_i \in A, \overline{ca_i} \leq \overline{ca_j}\}$*.*

Etant donné $\Omega \subseteq A$ (dans l'algorithme, systématiquement une coupe à gauche de A par une activité), l'enveloppe énergétique est utilisée pour calculer une

borne inférieure $lb(\overline{minc}(\Omega))$ pour la date de fin au plus tôt $\overline{minc}(\Omega)$ de Ω. D'après [13], $lb(\overline{minc}(\Omega)) = \lceil Env(\Omega)/capa \rceil$. Une fois les enveloppes énergétiques disponibles, il est possible de détecter les précédences.

Règle de précédence 1 *Si* $lb(\overline{minc}(LCut(A, a_j) \cup \{a_i\})) > \overline{ca_j}$, *alors* $LCut(A, a_j) \lessdot a_i$, *ce qui équivalent à : Si* $Env(LCut(A, a_j) \cup \{a_i\}) > Area(0, \overline{ca_j})$, *alors* $LCut(A, a_j) \lessdot a_i$.

A partir des précédences, l'algorithme de Vilìm met à jour les dates de début au plus tôt $\underline{sa_i}$ pour chaque activité qui finit nécessairement après la fin d'un ensemble d'activités Ω. Il est basé sur la notion de *compétition* : Ω *est en compétition* avec a_i ssi, $\underline{e}(\Omega) > Area(\underline{mins(\Omega)}, \overline{maxc(\Omega)}) - \underline{ra_i} * (\overline{maxc(\Omega)} - \underline{mins(\Omega)})$.

Règle de filtrage 2 *Si* $\Omega \lessdot a_i$ *et* Ω *est en compétition avec* a_i *alors* $[\underline{sa_i}, \underline{mins(\Omega)} + \left\lceil \frac{\underline{e}(\Omega) - Area(\underline{mins(\Omega)}, \overline{maxc(\Omega)}) + \underline{ra_i} * (\overline{maxc(\Omega)} - \underline{mins(\Omega)})}{\underline{ra_i}} \right\rceil [$ *peut être retiré de* $D(sa_i)$.

L'algorithme de Vilím utilise une structure de données arborescente pour calculer l'enveloppe énergétique $Env(\Omega)$ d'un ensemble donné $\Omega \subseteq A$, le Θ-tree, donnant une complexité temporelle globale de $O(k \cdot n \cdot log(n))$ où k est le nombre de $\underline{ra_i}$ distincts pour les activités a_i dans A.

4.2 Filtrage à partir des coûts maximaux

Des déductions peuvent être effectuées à partir d'événements sur les bornes supérieures de variables *Cost*. Les interactions avec la variable *obj* correspondant à des contraintes classiques de somme ou de max, nous nous focalisons sur le filtrage des variables de début.

4.2.1 Sweep pour SoftCumulative

Le filtrage ne dépend pas seulement de la capacité *capa*, mais également de la valeur maximale des variables *Cost*, et de *costC*. La notion de profil cumulatif est la même. Nous ajoutons une nouvelle classe d'événements : le début et la fin de chaque intervalle utilisateur $p_j \in P$. En conséquence, le rectangle défini par $\langle [\delta, \delta'[, sum_h \rangle$ a une capacité locale unique lc_j et correspond à une unique variable de coût $cost_j$. Nous avons deux propriétés : (1) Ces nouvelles activités ne modifient pas le calcul incrémental de sum_h dans le sweep. (2) Contrairement à la Règle de filtrage 1, la capacité maximale $capa_j$ à considérer pour un rectangle donné dépend de lc_j, $\overline{cost_j}$ et $costC$. Nous considérons d'abord $costC = max$.

Règle de filtrage 3 ($costC = max$) *Soit* $a_i \in ActToPrune$, *n'ayant pas de partie obligatoire enregistrée dans le rectangle* $\langle [\delta, \delta'[, sum_h \rangle$. *Si* $sum_h + \underline{ra_i} > lc_j + \overline{cost_j}$ *alors* $]\delta - \underline{da_i}, \delta'[$ *peut être retiré de* $D(sa_i)$.

Preuve 1 *Tout* a_i *dans* $ActToPrune$ *est tel qu'il existe au moins un point de temps* t *dans* $[\underline{sa_i}, \overline{ca_i}[$ *tel que* $\delta \leq t < \delta'$. *Par les Définitions 2 et 6, la hauteur* h_t *de toute solution étendant l'instantiation partielle courante est telle que* $h_t \geq sum_h + \underline{ra_i}$. *Alors, si* $sum_h + \underline{ra_i} - lc_j > \overline{cost_j}$, *la contrainte C3 de la Définition 4 est violée. Le même raisonnement s'applique en chaque point de temps de* $[\delta, \delta'[\cap [\underline{sa_i}, \overline{ca_i}[$. □

Si $costC = sum$, dans chaque intervalle utilisateur p_j, $cost_j$ correspond à l'aire dépassant la capacité locale lc_j dans cet intervalle. Rappelons que dans un intervalle donnée $[\delta, \delta'[$ défini dans l'algorithme de sweep, la capacité locale lc_j et la hauteur du profil sum_h restent constants par définition. La consommation du profil est $sum_h * (\delta' - \delta)$ et l'aire disponible totale ne menant pas à un dépassement est $lc_j * (\delta' - \delta)$. Nous essayons d'ajouter une activité a_i qui a une date de début au plus tôt $\underline{sa_i}$ dans l'intervalle $[\delta, \delta'[$. Si a_i a une durée minimum telle que a_i peut être contenue dans $[\delta, \delta'[$, alors son énergie minimale dans cet intervalle est $\underline{e}(a_i)$. Sinon, sa date de fin au plus tôt est à l'extérieur de $[\delta, \delta'[$, et son énergie dans cet intervalle est $(\delta' - \underline{sa_i}) * \underline{ra_i}$. Nous comparons l'énergie totale $(\delta' - \delta) * sum_h + \min((\delta' - \underline{sa_i}) * \underline{ra_i}, \underline{e}(a_i))$ à la ressource maximale disponible $lc_j * (\delta' - \delta) + \overline{cost_j}$. Nous filtrons $\underline{sa_i}$ si $\overline{cost_j}$ est dépassée.

Règle de filtrage 4 ($costC = sum$) *Soit* $a_i \in ActToPrune$, *n'ayant pas de partie obligatoire enregistrée dans le rectangle* $\langle [\delta, \delta'[, sum_h \rangle$ *et telle que* $\delta \leq \underline{sa_i} < \delta'$.
Si $(\delta' - \delta) * sum_h + \min((\delta' - \underline{sa_i}) * \underline{ra_i}, \underline{e}(a_i)) > lc_j * (\delta' - \delta) + \overline{cost_j}$ *alors* $[\underline{sa_i}, \min(\delta', \delta' -$

$$\lfloor (\overline{cost_j} + (lc_j - sum_h) * (\delta' - \delta)) / (\underline{ra_i}) \rfloor)[$$

peut être retiré de $D(sa_i)$.

Preuve 2 *Etant donnée* $a_i \in A$ *telle que* $\delta \leq \underline{sa_i} < \delta'$, *supposons que* a_i *commence en* $t = \underline{sa_i}$. *Si* $\underline{ca_i} < \delta'$, *sa contribution énergétique minimale est* $\underline{e}(a_i)$ *dans* $[\delta, \delta'[$, *sinon, c'est* $\underline{ra_i} * (\delta' - t)$. *Par les Définitions 2 et 6,* $(\delta' - \delta) * \underline{h_t} \geq (\delta' - \delta) * (sum_h + \underline{ra_i})$ *et* $(\delta' - \delta) * (sum_h + \underline{ra_i}) \geq (\delta' - \delta) * sum_h + \min((\delta' - t) * \underline{ra_i}, \underline{e}(a_i))$. *Alors, si* $(\delta' - \delta) * sum_h + \min((\delta' - t) * \underline{ra_i}, \underline{e}(a_i)) - lc_j * (\delta' - \delta) > \overline{cost_j}$ *(condition de filtrage), la contrainte C3 de la Définition 4 est violée.* $\overline{cost_j} + (lc_j - sum_h) * (\delta' - \delta)$ *est l'aire restant disponible et* $\lfloor \frac{\overline{cost_j} + (lc_j - sum_h) * (\delta' - \delta)}{\underline{ra_i}} \rfloor$ *est le nombre*

de points de temps pouvant être pris par a_i dans $[\delta, \delta'[$ sans violer la condition de filtrage. a_i ne peut pas commencer avant δ', ce qui mène à $\min(\delta', \delta' - \lfloor(\overline{cost_j} + (lc_j - sum_h) * (\delta' - \delta))/(\underline{ra_i})\rfloor)$. □

Le nombre d'événements dans le sweep dépend à la fois du nombre d'intervalles utilisateur et du nombre d'activités. La complexité temporelle est $O((n + p) \cdot log(n + p))$.

4.2.2 Edge-Finding pour SoftCumulative

Pour étendre l'algorithme de Vilìms au cas de la SoftCumulative, nous considérons les intervalles P et les capacités locales *Loc* au lieu d'une capacité unique *capa*. La Définition 7 (aire disponible) doit être adaptée. Dans un intervalle $I = [a, b[$, la ressource disponible dépend des capacités locales et des valeurs maximales des variables *Cost*. Nous devons étudier étudier le cas où a et b sont dans le même intervalle utilisateur et le cas où $p_j(a) \neq p_j(b)$.

Définition 9 (Aire disponible) *Etant donné un intervalle de temps $I = [a, b[$, nous notons $Area(a, b)$ la ressource maximale disponible. Nous distinguons deux cas :*

- *Si $costC = max$*
  - *Si $p_j(a) = p_j(b)$, $Area(a, b) = (b - a) * (lc_j(a) + \overline{cost_j}(a))$*
  - *Sinon, $Area(a, b) = (ep_j(a) - a) * (lc_j(a) + \overline{cost_j}(a)) + (b - sp_j(b)) * (lc_j(b) + \overline{cost_j}(b)) + \sum_{p_i \in]p_j(a), p_j(b)[} (ep_i - sp_i) * (lc_i + \overline{cost_i})$*
- *Si $costC = sum$*
  - *Si $p_j(a) = p_j(b)$, $Area(a, b) = (b - a) * lc_j(a) + \overline{cost_j}(a)$*
  - *Sinon, $Area(a, b) = (ep_j(a) - a) * lc_j(a) + \overline{cost_j}(a) + (b - sp_j(b)) * lc_j(b) + \overline{cost_j}(b) + \sum_{p_i \in]p_j(a), p_j(b)[} ((ep_i - sp_i) * lc_i + \overline{cost_i})$*

Définition 10 (Aire libre) *Etant donné un intervalle de temps $I = [a, b[$, nous notons $FreeArea(a, b)$ la ressource maximale disponible sans créer de nouvelle augmentation de dépassements. Le calcul est similaire à la Définition 9 excepté que les valeurs maximales des variables de coût ("$\overline{cost_j}$" et "$\overline{cost_i}$") sont remplacées par* 0.

Les principes pour calculer l'enveloppe énergétique et la coupe à gauche (voir les Définitions 7 et 8) restent les mêmes que dans le cas Cumulative, excepté que l'aire disponible est maintenant calculée grâce à la Définition 9. De plus, la Règle de précédence 1 reste également la même. Pour prouver ceci, nous introduisons la notation suivante (il n'y a pas besoin de calculer explicitement cette quantité dans l'algorithme).

Notation 2 (*inf*) *Etant donné Ω un sous-ensemble d'activités de A, et son enveloppe énergétique $Env(\Omega)$, $inf(\Omega)$ est le plus grand index j tel que $Area(sp_0, sp_j) \leq Env(\Omega)$.*

Comme dans le cas de Cumulative, étant donné un ensemble d'activités $\Omega \subseteq A$ (qui est, dans l'algorithme, systématiquement une coupe à gauche), nous adaptons le calcul d'une borne inférieure $lb(\overline{minc}(\Omega))$ pour la date de fin au plus tôt $\overline{minc}(\Omega)$ dans le cas où $costC = max$.[2] Rappelons que $sp_{inf(\Omega)}$ est le début de l'intervalle utilisateur d'index $inf(\Omega)$.

Proposition 1 $lb(\overline{minc}(\Omega)) = sp_{inf(\Omega)} + \left\lceil \frac{Env(\Omega) - Area(sp_0, sp_{inf(\Omega)})}{lc_{inf(\Omega)} + \overline{cost_{inf(\Omega)}}} \right\rceil$

Preuve 3 *$lb(\overline{minc}(\Omega))$ est le premier point de temps avant lequel $Env(\Omega)$ peut rentrer, en commençant en sp_0 (voir [13]). Par la Définition 2, $sp_{inf(\Omega)} \leq lb(\overline{minc}(\Omega)) < sp_{inf(\Omega)+1}$. $Env(\Omega) - Area(sp_0, sp_{inf(\Omega)})$ rentre dans $[sp_{inf(\Omega)}, sp_{inf(\Omega)+1}[$ en un nombre minimum de points de temps $\lceil (Env(\Omega) - Area(sp_0, sp_{inf(\Omega)})) / (lc_{inf(\Omega)} + \overline{cost_{inf(\Omega)}})\rceil$.* □

A partir de la Proposition 1, l'équivalence de la Règle de précédence 1 reste vraie.

Propriété 1 *Les deux conditions suivantes sont équivalentes : (1) $lb(\overline{minc}(LCut(A, a_j) \cup \{a_i\})) > \overline{ca_j}$. (2) $Env(LCut(A, a_j) \cup \{a_i\}) > Area(sp_0, \overline{ca_j})$.*

Preuve 4 *A partir de la Proposition 1 et de la Définition 7, avec $\Omega = LCut(A, a_j) \cup \{a_i\}$.* □

Avec le nouveau calcul de l'aire disponible, la notion de compétition et la Règle de filtrage 2 restent les mêmes qu'avec Cumulative. A chaque feuille du Θ-tree de Vilìm, l'aire disponible prend $O(p)$ au lieu de $O(1)$ (voir la Définition 9). Le calcul des noeuds internes reste le même que dans l'algorithme de Vilìm. Alors, la mise à jour des dates de début d'activités se fait en $O(n \cdot p)$, si on ajoute la détection des précédences, on a une complexité globale $O(p \cdot k \cdot n \cdot log(n))$.

4.3 Bornes inférieures des coûts et de l'objectif

Cette section montre comment on peut mettre à jour le minimum des variables de coût et comment nous pouvons calculer des bornes inférieures de l'objectif, à partir des domaines des variables d'activités. Ces bornes inférieures peuvent être utilisées pour mettre à jour $\underline{obj}$.

2. Nous donnons seulement la preuve dans le cas où $costC = max$. Dans le cas où $costC = sum$ la démonstration est la même, mis à part le calcul de $lb(\overline{minc}(\Omega))$ qui est égal à $lb(\overline{minc}(\Omega)) = sp_{inf(\Omega)} + \lceil \frac{Env(\Omega) - Area(sp_0, sp_{inf(\Omega)}) - \overline{cost_{inf(\Omega)}}}{lc_{inf(\Omega)}} \rceil$.

4.3.1 Dans l'algorithme de sweep

Les variables *Cost* peuvent être directement mises à jour dans l'algorithme de sweep, au cours du calcul du profil.

Règle de filtrage 5 *Considérons le rectangle courant* $\langle[\delta, \delta'[, sum_h\rangle$ *dans le sweep.*

- *Si* $(costC = max)$ *alors si* $sum_h - lc_j(\delta) > \underline{cost_j}(\delta)$ *alors* $[\underline{cost_j}(\delta), sum_h - lc_j(\delta)[$ *peut être retiré de* $D(\underline{cost_j(\delta)})$.
- *Si* $(costC = sum)$ *alors si* $(\delta' - \delta) * (sum_h - lc_j(\delta)) > \underline{cost_j}(\delta)$ *alors* $[\underline{cost_j}(\delta), (sum_h - lc_j(\delta)) * (\delta' - \delta)[$ *peut être retiré de* $D(cost_j(\delta))$.

Preuve 5 *A partir des Définitions 5 et 6.* □

Les bornes inférieures pour la variable objectif sont : si $objC = sum$ alors $LB = \sum_{j\in[0,k-1]} \underline{cost_j}$ et si $objC = max$ alors $LB = \max_{j\in[0,k-1]}(\underline{cost_j})$. Ces bornes inférieures peuvent être calculées incrémentalement sans augmentation de complexité temporelle.

4.3.2 Dans l'algorithme d'Edge-Finding

Les déductions sur les valeurs minimales des variables *Cost* en utilisant l'edge-finding sont faibles. De ce fait, nous nous concentrons uniquement sur le calcul d'une borne inférieure pour l'objectif.

Pour chaque activité $a_i \in A$, l'énergie minimale $\underline{e}(LCut(A, a_i))$ est calculée dans l'algorithme d'Edge-finding. Cette énergie devrait tenir dans l'aire définie par l'intervalle $[sp_0, \overline{ca_i}[$ par la Définition 8. Sinon, cela implique des dépassements et est susceptible de modifier la valeur minimale de la variable objectif. Suivant cette procédure, pour chaque activité, nous calculons une borne inférieure spécifique $LB(a_i)$ pour la variable objectif. Pour garder un calcul simple de chaque $LB(a_i)$, la proposition suivante considère qu'il n'y a pas de limite maximale sur le coût de chaque intervalle. Nous notons $\mathcal{P}$ un ensemble contenant tous les intervalles utilisateurs p_j tels que $sp_j \leq \overline{ca_i}$.

Proposition 2 *Soit* a_i *une activité de* A, *et* $LCut(A, a_i)$ *la coupe à gauche de* A *par* a_i. *Si* $\underline{e}(LCut(A, a_i)) > FreeArea(sp_0, \overline{ca_i})$ *alors*

- *Si* $(costC = max$ *et* $objC = sum)$ *alors* $LB(a_i) = \lceil \frac{\underline{e}(LCut(A,a_i)) - FreeArea(sp_0,\overline{ca_i})}{\max_{p_j\in\mathcal{P}}(min(\overline{ca_i}, sp_{j+1}) - sp_j))} \rceil$.
- *Si* $(costC = max$ *et* $objC = max)$ *alors* $LB(a_i) = \lceil \frac{\underline{e}(LCut(A,a_i)) - FreeArea(sp_0,\overline{ca_i})}{\overline{ca_i} - sp_0} \rceil$.
- *Si* $(costC = sum$ *et* $objC = sum)$ *alors* $LB(a_i) = \underline{e}(LCut(A, a_i)) - FreeArea(sp_0, \overline{ca_i})$.
- *Si* $(costC = sum$ *et* $objC = max)$ *alors* $LB(a_i) = \lceil \frac{\underline{e}(LCut(A,a_i)) - FreeArea(sp_0,\overline{ca_i})}{|\mathcal{P}|} \rceil$.

Preuve 6 *Par la Définition 8,* $\underline{e}(LCut(A, a_i))$ *est l'énergie minimale nécessairement placée avant* $\overline{ca_i}$. *L'énergie maximale rentrant dans* $[sp_0, \overline{ca_i}[$ *sans mener à un dépassement est* $FreeArea(sp_0, \overline{ca_i})$ *par la Définition 10. De ce fait, si* $\underline{e}(LCut(A, a_i)) > FreeArea(sp_0, \overline{ca_i})$ *alors cela induit une borne inférieure sur* $\underline{obj}$. *nous considérons* $costC = max$ *et* $objC = sum$. *Sans perte de généralité, nous considérons que toute l'aire de dépassement peut être placée dans l'intervalle de longueur maximale* I *commençant en* sp_j, *c'est à dire que nous considérons* $\max_{p_j\in\mathcal{P}}(min(\overline{ca_i}, sp_{j+1}) - sp_j)$ *: si l'on considère un ou plusieurs intervalles additionnels ayant, par définition, une longueur plus petite ou égale à la longueur de* I, *cela mènerait nécessairement à une somme de dépassements supérieure ou égale (rappelons que nous ne considérons pas de limite supérieure sur les coûts).* $LB(a_i)$ *est égal à* $\lceil \frac{\underline{e}(LCut(A,a_i)) - FreeArea(sp_0,\overline{ca_i})}{\max_{p_j\in\mathcal{P}}(min(\overline{ca_i}, sp_{j+1}) - sp_j))} \rceil$. *Si* $costC = max$ *et* $objC = max$ *alors nous étalons l'aire de dépassement sur le plus grand intervalle possible, c'est à dire* $[sp_0, \overline{ca_i}[$. *Si* $costC = sum$ *et* $objC = sum$, *la borne inférieure est la même quelle que soit la répartition des dépassements entre les intervalles utilisateur. Si* $costC = sum$ *et* $objC = max$, *en étalant le dépassement* $\underline{e}(LCut(A, a_i)) - FreeArea(sp_0, \overline{ca_i})$ *sur les intervalles* $\mathcal{P}$ *on obtient le plus petit coût maximum possible.* □

Notons que considérer les limites maximales sur les coûts ne peut qu'augmenter les bornes inférieures calculées. Le calcul de $LB(a_i)$ est intégré dans l'algorithme d'edge-finding et est effectué en $O(p)$. Alors, la complexité globale en intégrant cet algorithme dans l'edge-finding est toujours $O(k \cdot p \cdot n \cdot log(n))$.

4.4 Intégration des bornes inférieures de l'objectif

Dans cette section, nous considérons que les procédure de filtrage 4.2.1 ont été appliquées. Notre but est de filtrer les variables d'activités en utilisant les bornes inférieures de la variable objectif calculées dans la Section 4.3. Nous effectuons cette intégration en deux phases. La première est une seconde procédure de balayage, tandis que la seconde est un raisonnement énergétique. Sans perte de généralité, nous considérons que $\underline{obj} = LB$.

4.4.1 Sweep

Dans cette section, nous utilisons la borne inférieure LB de l'objectif calculée dans la Section 4.3.1. Nous considérons des activités n'ayant pas de partie obligatoire dans le rectangle courant $\langle[\delta, \delta'[, sum_h\rangle$, et la capacité locale correspondante lc_j.

Rappelons que LB et $\underline{cost_j}$ ont été calculées dans la première phase de balayage. Alors, si $costC = max$ alors $\underline{cost_j} \geq max(sum_h - lc_j, 0)$, et si $costC = sum$ alors $\underline{cost_j} \geq (\delta' - \delta) * max(sum_h - lc_j, 0)$.

Nous essayons d'ajouter une activité a_i dans l'intervalle $[\delta, \delta'[$, et ensuite nous calculons le coût obtenu en considérant que a_i est programmée dans $[\delta, \delta'[$. Si le coût calculé est strictement supérieur à $\underline{cost_j}$, alors nous pouvons considérer une LB augmentée, sous condition que a_i soit placée dans $[\delta, \delta'[$.

Définition 11 *Soit $a_i \in ActToPrune$, n'ayant pas de partie obligatoire enregistrée dans le rectangle $\langle[\delta, \delta'[, sum_h\rangle$, et p_j l'unique intervalle contenant $[\delta, \delta'[$. Nous définissons $cost_j^{a_i}$ tel que :*

- *Si($costC = max$) alors $cost_j^{a_i} = max(sum_h + \underline{ra_i} - lc_j, 0)$.*
- *Si($costC = sum$) alors nous ne considérons que les activités a_i telles que $\delta \leq \underline{sa_i} < \delta'$ et $cost_j^{a_i} = max((\delta' - \delta) * (sum_h - lc_j) + \min((\delta' - \underline{sa_i}) * \underline{ra_i}, \underline{e}(a_i)), 0)$.*

Règle de filtrage 6 *Soit $a_i \in ActToPrune$, n'ayant pas de partie obligatoire enregistrée dans le rectangle $\langle[\delta, \delta'[, sum_h\rangle$, et p_j l'unique intervalle contenant $[\delta, \delta'[$.*

- *Si($costC = max$ et $objC = sum$) alors si $LB + max(cost_j^{a_i} - \underline{cost_j}, 0) > \overline{obj}$ alors $]\delta - \underline{da_i}, \delta'[$ peut être retiré de $D(sa_i)$.*
- *Si$costC = max$ et $objC = max$) alors si $cost_j^{a_i} > \overline{obj}$ alors $]\delta - \underline{da_i}, \delta'[$ peut être retiré de $D(sa_i)$.*
- *Si($costC = sum$ et $objC = sum$) alors si $LB + max(cost_j^{a_i} - \underline{cost_j}, 0) > \overline{obj}$ alors $[\underline{sa_i}, \min(\delta', \delta' - \left\lfloor \frac{(\overline{obj} - LB + \underline{cost_j}) + (lc_j - sum_h) * (\delta' - \delta)}{\underline{ra_i}} \right\rfloor)[$ peut être retiré de $D(sa_i)$.*
- *Si($costC = sum$ et $objC = max$) alors si $cost_j^{a_i} > \overline{obj}$ alors $[\underline{sa_i}, \min(\delta', \delta' - \left\lfloor \frac{\overline{obj} + (lc_j - sum_h) * (\delta' - \delta)}{\underline{ra_i}} \right\rfloor)[$ peut être retiré de $D(sa_i)$.*

Preuve 7 *$cost_j^{a_i}$ représente le coût minimal dans un intervalle $[\delta, \delta'[\subseteq p_j$ si nous ajoutons l'activité a_i à $[\delta, \delta'[$. Par la Définition 4, l'augmentation dans LB est $max(cost_j^{a_i} - \underline{cost_j}, 0)$ si $objC = sum$. Si cette augmentation est supérieure à la marge autorisée par $\overline{obj}$, alors nous pouvons filtrer sa_i. Si $costC = max$, tout point de temps pris par a_i dans $[\delta, \delta'[$ peut être responsable de l'augmentation, donc $]\delta - \underline{da_i}, \delta'[$ peut être retiré de $D(sa_i)$. Si $costC = sum$, le $cost_j$ maximal autorisé par $\overline{obj}$ est $\overline{obj} - LB + \underline{cost_j}$. Nous filtrons ensuite l'intervalle de la même manière que dans la Règle de filtrage 4. Le raisonnement est similaire pour $objC = max$ sauf que nous ne mesurons pas une augmentation, mais nous comparons directement le coût induit avec $\overline{obj}$.* □

4.4.2 Raisonnement énergétique

L'algorithme d'Edge-Finding calcule, pour chaque activité $a_i \in A$, l'énergie minimale $\underline{e}(LCut(A, a_i))$. Nous considérons ensuite la borne inférieure $LB(a_i)$ de la variable objectif calculée dans la Section 4.3.2.

Nous essayons d'ajouter une activité a_j telle que $\underline{sa_j} < \overline{ca_i}$ et $a_j \notin LCut(A, a_i)$ dans l'intervalle $[sp_0, \overline{ca_i}[$. Nous pouvons ensuite calculer l'énergie totale obtenue en considérant que a_j est placée dans l'intervalle $[sp_0, \overline{ca_i}[$. Si l'énergie calculée moins l'aire disponible dans cet intervalle est strictement plus grande que $LB(a_i)$, alors nous pouvons considérer une $LB(a_i)$ augmentée sous condition que a_i soit placée dans $[sp_0, \overline{ca_i}[$. Alors si cette borne inférieure est strictement plus grande que $\overline{obj}$, nous pouvons filtrer sa_i.

Définition 12 *Soient a_i and a_j deux activités telles que $sp_0 \leq \underline{sa_j} < \overline{ca_i}$. Nous définissons $e(a_i, a_j)$ comme l'énergie de a_j dans cet intervalle sous condition que a_j commence en $\underline{sa_j}$: $e(a_i, a_j) = min(\underline{e}(a_j), (\overline{ca_i} - \underline{sa_j}) * \underline{ra_j})$.*

Nous nous concentrons ici sur $costC = max$ et $objC = sum$. Les autres cas sont similaires.

Règle de filtrage 7 ($costC = max$ et $objC = sum$) *Soient a_i et a_j deux activités dans A telles que $\underline{sa_j} < \overline{ca_i}$ et $a_j \notin LCut(A, a_i)$, et $LB(a_i)$ la borne inférieure de obj calculée dans la Section 4.3.2. Si $LB(a_i) + \lceil \frac{e(a_i, a_j)}{\max_{p_k \in \mathcal{P}}(min(\overline{ca_i}, sp_{k+1}) - sp_k))} \rceil > \overline{obj}$ alors $[\underline{sa_j}, min(\overline{ca_i}, \overline{ca_i} - \lfloor \frac{(\overline{obj} - LB(a_i)) * (\max_{p_j \in \mathcal{P}}(min(\overline{ca_i}, sp_{j+1}) - sp_j))}{\underline{ra_j}} \rfloor[$ peut être retiré de $D(sa_j)$.*

Preuve 8 *$LB(a_i)$ ignore $a_j \notin LCut(A, a_i)$. A partir de la Proposition 2, la règle est vérifiée.* □

4.5 Filtrage à partir des minimums de coûts

Dans la plupart des cas, en PPC, nous filtrons les activités à partir des bornes supérieures des variables de coût. La Section 3 montre qu'effectuer des déductions à partir des augmentations sur les minimums des domaines des variables *Cost* peut avoir un intérêt. Nous introduisons une technique basée sur la notion de partie enveloppante qui est duale à la notion de partie obligatoire.

Définition 13 (Partie enveloppante). *La* partie enveloppante $ep(a_i)$ *d'une activité* $a_i \in A$*, est l'union de tous les placements possibles pour* a_i*. Elle est définie par l'intervalle* $[\underline{sa_i}, \overline{ca_i}[$ *et une hauteur égale à* $\overline{ra_i}$ *sur* $[\underline{sa_i}, \overline{ca_i}[$*, et nulle ailleurs.*

(Profil cumulatif enveloppant). *Le* profil cumulatif enveloppant $EnvP$ *est la consommation de ressource cumulée maximale, dans le temps, de toutes les activités. Pour un point de temps donné* t*, la hauteur de* $EnvP$ *en* t *est égale à* $\sum_{a_i \in A, t \in [\underline{sa_i}, \overline{ca_i}[} \overline{ra_i}$ *(somme des contributions de toutes les parties enveloppantes passant par* t*).*

Pour filtrer les activités à partir des valeurs minimales des variables de coût, nous appliquons une procédure de balayage sur le profil cumulatif enveloppant et sur le profil des parties obligatoires. Pour ce faire, nous proposons d'ajouter une nouvelle classe d'événements : le début et la fin de la partie enveloppante de chaque activité. Ces nouveaux événements ne modifient pas le calcul incrémental de sum_h dans le sweep. De plus, nous considérons une nouvelle hauteur sum_e, qui est la hauteur du profil cumulatif enveloppant dans le rectangle courant. Intuitivement, le principe du filtrage est le suivant : Pendant la construction de $EnvP$, à chaque événement égal au début d'un intervalle utilisateur sp_j nous vérifions si il y a un unique rectangle $\langle[\delta, \delta'[, sum_e\rangle$ dans $EnvP$ entre sp_j et ep_j tel que $sum_e \geq lc_j + \underline{cost_j}$. Si c'est vrai, nous filtrons $a_i \in A$ avec la règle suivante.

1. Retirer de $EnvP$ la contribution de a_i.
2. Filtrer les bornes de $D(sa_i)$ pour s'assurer que si a_i commence en $\underline{sa_i}$ ou $\overline{sa_i}$ alors à tout point de temps t entre 0 et m, $EnvP$ peut atteindre $\underline{cost_j(t)}$.

Règle de filtrage 8 ($costC = max$) *Considérons un rectangle* $\langle[\delta, \delta'[, sum_e\rangle$ *et un intervalle* p_j *tel que* $\langle[\delta, \delta'[, sum_e\rangle$ *soit l'unique rectangle satisfaisant* $sum_e \geq lc_j + \underline{cost_j}$*. Soit* $a_i \in ActToPrune$*, si* $sum_e - \overline{ra_i} < lc_j + \underline{cost_j}$ *alors :* $[\underline{sa_i}, \delta - \overline{da_i}]$ *et* $[\delta', \overline{sa_i}]$ *peuvent être retiré de* $D(sa_i)$*.*

Preuve 9 *Si* $sum_e - \overline{ra_i} < lc_j + \underline{cost_j}$ *alors* a_i *devrait intersecter* $[\delta, \delta'[$*, sinon dans toute solution étendant l'instantiation partielle courante, aucun point de temps* t *dans* $[\delta, \delta'[$ *ne satisfera* $h_t \geq lc_j + cost_j$*. La contrainte C3 de la Définition 4 sera violée.* □

Nous considérons que, pour chaque événement correspondant au début d'un intervalle utilisateur, nous essayons d'applique la Règle de filtrage 8 par rapport à l'intervalle utilisateur précédent. Cela peut être effectué en $O(1)$. Nous ajoutons $O(n)$ événements au sweep et l'existence et la position de l'unique rectangle $\langle[\delta, \delta'[, sum_e\rangle$ satisfaisant $sum_e \geq lc_j + \underline{cost_j}$ peuvent être maintenus en $O(1)$. La complexité globale ne change pas.

5 Stratégies de recherche et expérimentation

Nous avons conçu SCStrategy, une stratégie de recherche spécifique au problème représenté par notre contrainte. Elle est basée sur deux règles de priorité. (1) Choix de variable : sélectionner l'ensemble des activités maximisant la hauteur, et parmi elles sélectionner la variable de début d'une activité maximisant la durée. (2) Choix de valeur : Etant donnée la variable choisie, calculer l'ensemble des points de temps minimisant l'augmentation d'objectif, grâce à un balayage. Trier cet ensemble en fonction de l'énergie laissée libre une fois l'activité ajoutée en ce point, dans un ordre croissant

Nous avons implémenté SoftCumulative dans choco [1], avec $costC = max$ et $objC = sum$. Nous avons généré aléatoirement des instances avec des activités de durées 1 à 4 et de hauteurs 1 à 3, des capacités locales entre 3 et 6 ainsi que des longueurs pour les intervalles utilisateur. Les coûts maximaux ont été fixés à 6. Chaque activité peut être placée entre 0 et l'horizon. Les résultats dans les tableaux sont des moyennes sur 50 instances pour chaque classe de problème, en utilisant un processeur Intel Core 2, avec 4 Go de RAM.

Le premier test évalue le passage à l'échelle de notre contrainte, avec 16 ou 32 intervalles utilisateur. Avec notre implémentation, utiliser uniquement l'algorithme de sweep pour trouver une première solution est l'approche la plus efficace par rapport au temps. La première table (au dessus) donne les résultats obtenus en débranchant l'algorithme d'edge-finding, et montre que SoftCumulative peut être utilisée pour trouver un solution pour des problèmes impliquant 1000 activités. SCStrategy donne généralement, et plus rapidement, une solution avec une meilleure valeur d'objectif ($\pm 20\%$), comparé à une heuristique assignant statiquement les variables de début avec une valeur aléatoire (la meilleure heuristique par défaut dans choco avec nos instances). Comme la complexité temporelle du sweep dépend du nombre d'intervalles, trouver une solution prend plus de temps lorsqu'il y a plus d'intervalles.

Dans un second benchmark nous essayons de trouver des solutions optimales. La deuxième table montre clairement que, dans ce cas, utiliser l'edge-finding améliore le processus de résolution. Les instances impliquent 4 ou 8 intervalles utilisateur et des contraintes

# activités / # intervalles utilisateur	longueur max des intervalles utilisateur	# résolus (avec le meilleur *obj*)		temps moyen (# backtracks)	
		Random	SCSTRATEGY	Random	SCSTRATEGY
500 / 16	40	48 (1)	49 (47)	2.2s (51)	2.1s (46)
500 / 32	20	50 (0)	50 (50)	3.5s (59)	2.8s (11)
1000 / 16	80	49 (1)	39 (38)	31.7s (221)	22.4s (150)
1000 / 32	40	49 (1)	46 (45)	67.3s (382)	49.5s (136)

# activités / # intervalles utilisateur	longueur max des intervalles utilisateur	# prouvés infaisables		# résolus avec $obj = 0$		# résolus avec $obj \neq 0$		temps moyen (#backtracks) avec $obj \neq 0$	
		No EF	EF	No EF	EF	No EF	EF	No EF	EF
50 / 4	20	0	10	11	11	0	27	> 2mn	1.0s (2183)
50 / 8	10	0	6	9	9	0	32	> 2mn	8.1s (7512)
100 / 4	41	0	12	8	8	0	27	> 2mn	4.9s (5534)
100 / 8	21	0	2	8	8	0	33	> 2mn	21.8s (11494)
125 / 4	51	0	6	8	8	0	34	> 2mn	9.6s (8540)
125 / 8	25	0	2	5	5	0	36	> 2mn	47.5s (19058)

TABLE 1 – Au-dessus : temps et nombre de backtracks moyens pour trouver une première solution. Comparaison d'une stratégie Random avec SCSTRATEGY. # résolus est le nombre d'instances résolues avec, entre parenthèses, le nombre d'instances où la variable objectif est la plus petite parmi les instances qui peuvent être résolues par les deux stratégies. En-dessous : temps et nombre de backtracks moyens pour trouver une solution optimale avec des contraintes additionnelles. Concernant le nombre d'instances prouvées infaisables et résolues, "EF" signifie que nous avons utilisé l'algorithme d'edge-finding, et "No EF" signifie que nous l'avons débranché.

additionnelles. Pour définir ces contraintes, nous avons partitionné les coûts par classe de 4 variables, sur lesquelles nous avons imposé : (1) Au moins une variable de coût prend la valeur 0 dans chaque classe de la partition, et (2) $\forall i$ tel que $0 \leq i < |Cost| - 1$, $|cost_{i+1} - cost_i| \leq 2$. Les instances dans la deuxième table ont été testées en utilisant l'heuristique par défaut dans CHOCO, SCSTRATEGY n'étant pas adaptée pour prouver l'optimum. Cette deuxième table montre qu'en utilisant l'Edge-Finding nous pouvons trouver la solution optimale de problèmes comportant 125 activités.

6 Conclusion

Nous avons présenté une nouvelle contrainte pour résoudre des problèmes cumulatifs avec dépassements. Nous avons adapté un algorithme de sweep et l'algorithme d'edge-finding de Vilìm à notre contexte. Nous avons proposé une procédure de filtrage spécifique aux dépassements imposés de l'extérieur. Nous avons conçu une heuristique de recherche dédiée. Une de nos perspectives est de considérer également la relaxation des relations de précédence entre activités.

Références

[1] Choco : An open source Java CP library. `http://choco.mines-nantes.fr`, 2011.

[2] A. Aggoun and N. Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathl. Comput. Modelling*, 17(7) :57–73, 1993.

[3] N. Beldiceanu and M. Carlsson. Revisiting the cardinality operator and introducing the cardinality-path constraint family. *Proc. ICLP*, 2237 :59–73, 2001.

[4] N. Beldiceanu and M. Carlsson. A new multi-resource *cumulatives* constraint with negative heights. *Proc. CP*, pages 63–79, 2002.

[5] E. Hebrard, B. O'Sullivan, and I. Razgon. A soft constraint of equality : Complexity and approximability. *Proc. CP*, 5202 :358–371, 2008.

[6] A. Lahrichi. The notions of Hump, Compulsory Part and their use in Cumulative Problems. *C.R. Acad. sc.*, t. 294 :209–211, 1982.

[7] J. Larrosa and T. Schiex. Solving weighted csp by maintaining arc consistency. *Artificial Intelligence*, 159(1-2) :1–26, 2004.

[8] G. Pesant and J.-C. Régin. Spread : A balancing constraint based on statistics. *Proc. CP*, pages 460–474, 2005.

[9] T. Petit and E. Poder. Global propagation of side constraints for solving over-constrained problems. *Annals OR*, 184(1) :295–314, 2011.

[10] T. Petit and J-C. Régin. The ordered distribute constraint. In *ICTAI*, pages 431–438, 2010.

[11] T. Petit, J-C. Régin, and C. Bessière. Meta constraints on violations for over constrained problems. *Proc. IEEE-ICTAI*, pages 358–365, 2000.

[12] P. Schaus, Y. Deville, P. Dupont, and J-C. Régin. The deviation constraint. *Proc. CPAIOR*, 4510 :260–274, 2007.

[13] P. Vilím. Edge finding filtering algorithm for discrete cumulative resources in $O(kn\log(n))$. *Proc. CP*, pages 802–816, 2009.