Chapitre 8

Minimisation du retard algébrique maximal sur une machine à traitement par fournées

Ce chapitre présente une approche en programmation par contraintes pour le problème de minimisation du retard algébrique maximal sur une machine à traitement par fournées (voir chapitre 5). Le modèle proposé exploite une décomposition du problème : rechercher un placement réalisable des tâches dans les fournées ; minimiser le retard algébrique maximal des fournées sur une machine. Ce modèle est enrichi par une nouvelle contrainte globale pour l'optimisation qui utilise une relaxation du problème pour appliquer des techniques de filtrage basé sur les coûts. Les résultats expérimentaux démontrent l'efficacité de ces dernières. Des comparaisons avec d'autres approches exactes montrent clairement l'intérêt de notre approche : notre contrainte globale permet de résoudre des instances dont la taille est supérieur d'un ordre de grandeur par rapport à une formulation en programmation mathématique ou à un branch-and-price.

Sommaire

DANS ce chapitre, nous présentons nos travaux [8, 9] portant sur la minimisation du retard algébrique maximal d'un nombre fini de tâches de tailles différentes réalisées sur une machine à traitement par fournées. Une machine à traitement par fournées peut exécuter simultanément plusieurs tâches regroupées dans une fournée. Ces machines constituent souvent des goulots d'étranglement pour la planification de la production à cause des temps d'exécution élevés des tâches sur ce type de machine.
Nous nous intéressons dans ce chapitre plus particulièrement au problème $1|p\text{-}batch; b < n; non\text{-}identical|L_{max}$ introduit au chapitre 5. On considère un ensemble J de n tâches et une machine à traitement par fournées de capacité b. Chaque tâche j est caractérisée par un triplet d'entiers positifs (p_j, d_j, s_j), où p_j est sa durée

d'exécution, d_j est sa date échue, et s_j est sa taille. La machine à traitement par fournées peut traiter plusieurs tâches simultanément tant que la somme de leurs tailles ne dépasse pas sa capacité b. Toutes les données sont déterministes et connues a priori. La date de fin C_j d'une tâche est la date de fin de la fournée à laquelle elle appartient. Le critère d'optimalité étudié est la minimisation du retard algébrique maximal : $L_{max} = \max_{1 \leq j \leq n}(C_j - d_j)$. Ce problème est NP-difficile *au sens fort* puisque Brucker *et al.* [124] ont prouvé que le même problème avec des tâches de tailles identiques était NP-difficile *au sens fort*.
Ce chapitre est organisé de la manière suivante. La section 8.1 introduit notre modèle en programmation par contraintes. La section 8.2 décrit une nouvelle contrainte globale pour l'optimisation basée sur la résolution d'une relaxation du problème original qui intègre des techniques de filtrage basé sur les coûts. L'algorithme de recherche inspiré par des stratégies classiques en placement à une dimension utilisant une nouvelle heuristique de sélection de valeur est décrit en section 8.3. Finalement, la section 8.4 évalue les performances des composantes de notre approche avant de la comparer à une formulation en programmation mathématique et à une approche par *branch-and-price*.
Nous supposons dans le reste de ce chapitre que les tâches sont triées par taille décroissante ($s_j \geq s_{j+1}$) et que la taille de chaque tâche est inférieure à la capacité de la machine ($s_j \leq b$). On peut noter de plus que le nombre de fournées m est inférieur au nombre de tâches n.

8.1 Modèle en programmation par contraintes

Notre modèle est basé sur une décomposition du problème en un problème de placement des tâches dans les fournées et un problème de minimisation du retard algébrique maximal des fournées sur une machine. Le problème du placement des tâches dans les fournées est équivalent à un problème de placement à une dimension, connu pour être NP-complet (*bin packing*, voir section 3.5). L'énoncé de ce problème est le suivant : soit n articles (tâches) caractérisés chacun par une taille positive $s_j \geq 0$ et m conteneurs (fournées) caractérisés par une capacité b, existe-t'il un placement des n articles dans les m conteneurs tel que la somme des tailles des articles dans chaque conteneur reste inférieure à la capacité b ? Une fois que les tâches ont été placées dans les fournées, l'ordonnancement des fournées doit minimiser le retard algébrique maximal des fournées sur une machine. Ce problème, noté $1||L_{max}$, peut être résolu en temps polynomial [153] : un ordonnancement optimal est obtenu en appliquant la règle de Jackson, aussi appelée règle EDD (*earliest due date (EDD-)rule*) qui séquence les tâches en ordre non décroissant de leur date échue [154].

Nous décrivons maintenant le modèle PPC du problème étudié. Soit $J = [1, n]$ l'ensemble des indices des tâches et $K = [1, m]$ l'ensemble des indices des fournées. La valeur maximale des dates échues est notée $d_{max} = \max_J\{d_j\}$ et celle des durées d'exécution est notée $p_{max} = \max_J\{p_j\}$. La variable $B_j \in K$ représente l'indice de la fournée dans laquelle est placée la tâche j, et la variable $J_k \subseteq J$ représente l'ensemble des tâches contenues dans la fournée k. Ces deux ensembles de variables satisfont les contraintes de liaison : $\forall j \in J,\ B_j = k \Leftrightarrow j \in J_k$. Les variables entières positives $P_k \in [0, p_{max}]$, $D_k \in [0, d_{max}]$ et $S_k \in [0, b]$ représentent respectivement la durée d'exécution, la date échue, et la charge de la fournée k. Finalement, les variables $M \in [0, m]$ et L_{max} (non bornée) représentent respectivement le nombre de fournées non vides et l'objectif. Le modèle en programmation par contraintes est le suivant :

$$\texttt{maxOfASet}\left(P_k,\ J_k,\ [p_j]_J,\ 0\right) \qquad \forall k \in K \qquad (8.1)$$

$$\texttt{minOfASet}\left(D_k,\ J_k,\ [d_j]_J,\ d_{max}\right) \qquad \forall k \in K \qquad (8.2)$$

$$\texttt{pack}\left([J_k]_K,\ [B_j]_J,\ [S_k]_K,\ M,\ [s_j]_J\right) \qquad (8.3)$$

$$\texttt{sequenceEDD}\left([B_j]_J,\ [D_k]_K,\ [P_k]_K,\ M,\ L_{max}\right) \qquad (8.4)$$

Les contraintes (8.1), où $[p_j]_J$ est le tableau contenant les durées d'exécution, impose que la durée P_k d'une fournée k est la plus grande durée d'exécution de ses tâches (ensemble J_k) si la fournée n'est pas vide, et égale à 0 sinon. De la même manière, les contraintes (8.2) imposent que la date échue D_k de la fournée k est la plus petite date échue minimale de ses tâches, et égale à d_{max} sinon. En effet, le retard algébrique d'une fournée k défini par la relation $\max_{j \in J_k}(C_k - d_j)$ est égal à $C_k - \min_{j \in J_k}(d_j)$ où C_k est la date d'achèvement de la fournée k. Remarquez que le retard algébrique d'une tâche peut être négatif.

La contrainte (8.3) est une variante de la contrainte globale proposée par Shaw [70] pour le problème de *bin packing*. La contrainte applique des règles de filtrage inspirées de raisonnements sur les problèmes de sac à dos ainsi qu'une borne inférieure dynamique sur le nombre minimal de conteneurs non vides. Elle remplace les contraintes de liaison entre les variables représentant le placement des tâches dans les fournées ($\forall j \in J,\ B_j = k \Leftrightarrow j \in J_k$) et assure la cohérence entre le placement des tâches et les charges des conteneurs ($\forall k \in K,\ \sum_{j \in J_k} s_j = S_k$). De plus, `pack` réduit les domaines en tenant compte d'une contrainte redondante assurant la cohérence entre les charges des conteneurs et les tailles des tâches ($\sum_J s_j = \sum_K S_k$). Remarquez que le respect de la capacité de la machine est assuré par le domaine initial des variables S_k. Notre implémentation de la contrainte `pack` est décrite dans la section 9.3.9.
La contrainte globale (8.4) impose que la valeur de la variable objectif L_{max} soit égale au retard algébrique maximal des fournées lorsqu'elles sont ordonnancées en suivant la règle EDD. Cette contrainte applique plusieurs règles de filtrages détaillées dans la section 8.2.
Pour conclure, une solution de notre modèle PPC est composée d'un placement réalisable des tâches dans les fournées, et du retard algébrique maximal associé à l'instance du problème $1||L_{max}$ engendrée par les fournées. Remarquez que ce modèle reste valide en présence de contraintes additionnelles imposant des capacités non identiques des conteneurs, des incompatibilités entre les tâches, ou une répartition équilibrée de la charge des conteneurs. En effet, il suffit de modifier les domaines des variables associées pour prendre en compte ces nouvelles contraintes.

Korf [69] a observé que la prise en compte de conditions de dominance entre les placements réalisables, qui permettent de n'en considérer qu'un nombre restreint, est un élément crucial pour la résolution des problèmes de placement à une dimension. Les méthodes efficaces pour ces problèmes (voir section 3.5) utilisent intensivement des conditions d'équivalence et de dominance pour réduire l'espace de recherche. Mais, ces procédures considèrent généralement que les articles avec des tailles identiques ou les conteneurs avec des charges identiques peuvent être permutés sans dégrader la qualité de la solution. Dans le contexte d'une machine à traitement par fournées, une permutation entre deux tâches de tailles identiques peut modifier la qualité de la solution, car cette permutation peut entraîner une modification des durées d'exécution ou des dates échues des fournées. Par conséquent, ces règles ne peuvent pas être appliquées dans ce contexte. De la même manière, les méthodes par *bin completion* [66, 69, 71] ne peuvent pas être appliquées, car elles utilisent des conditions de dominance en considérant uniquement la charge des conteneurs.
Par contre, on peut toujours réduire l'espace de recherche en imposant que deux tâches i et j telles que $s_i + s_j > b$ appartiennent à des fournées différentes. Puisque les tâches sont triées par taille décroissante, notons j_0 le plus grand indice tel que chaque paire de tâches dont les indices sont inférieurs à j_0 appartiennent obligatoirement à des fournées différentes : $j_0 = \max\{j \mid \forall\ 1 \leq i < j,\ s_{i+1} + s_i > b\}$. Les contraintes (8.5), rangeant consécutivement les plus grandes tâches dans les premiers conteneurs, peuvent être ajoutées au modèle.

$$B_j = j \qquad 1 \leq j \leq j_0 \tag{8.5}$$

8.2 Description de la contrainte globale `sequenceEDD`

Les réductions de domaine proviennent généralement de raisonnements liés à la satisfiabilité. Dans le contexte de l'optimisation, les réductions de domaine peuvent aussi être réalisées en se basant sur des raisonnements liés à l'optimalité. Le filtrage tend alors à retirer des combinaisons de valeurs qui ne peuvent pas être impliquées dans une solution améliorant la meilleure solution découverte jusqu'à présent. Focacci *et al.* [155] ont proposé d'inclure ces raisonnements dans une contrainte globale qui représente une relaxation d'un sous-problème ou d'elle-même. Des composants internes à la contrainte globale fournissent des algorithmes de recherche opérationnelle calculant la solution optimale d'une relaxation du problème ainsi qu'une fonction estimant le gradient (variation) du coût de cette solution optimale après l'affectation d'une valeur à une variable. Focacci *et al.* ont montré l'intérêt d'utiliser cette information pour le filtrage, mais aussi pour guider la recherche sur plusieurs problèmes d'optimisation combinatoire.

Comme mentionné précédemment, la contrainte globale `sequenceEDD` assure que la valeur de la variable L_{max} soit le plus grand retard algébrique de la séquence de fournées ordonnancées en suivant la règle EDD. Cette contrainte utilise une relaxation du problème original fournissant une borne inférieure sur la valeur de la variable objectif pour réduire l'espace de recherche. L'idée principale est de déduire des

contraintes primitives à partir des informations sur les coûts des affectations. Un premier composant de la contrainte globale fournit donc une solution optimale pour une relaxation du problème qui consiste à minimiser le retard algébrique maximal des fournées sur une machine. Un autre composant est une fonction gradient donnant une estimation (borne inférieure) du coût à ajouter à celui de la solution optimale lors de l'affectation d'une valeur à certaines variables. Ainsi, la solution optimale de la relaxation permet d'améliorer la borne inférieure de la variable objectif, puis d'éliminer des parties de l'espace de recherche pour lesquelles les bornes inférieures sont supérieures à la meilleure solution découverte jusqu'à présent. La relaxation du problème est présentée dans la section 8.2.1 suivie de quatre règles de filtrage présentées dans les sections 8.2.2, 8.2.3 and 8.2.4.

8.2.1 Relaxation du problème

Dans cette section, nous décrivons la construction d'une instance $I(\mathcal{A})$ de la relaxation, c'est-à-dire une instance du problème $1||L_{max}$, à partir d'une affectation partielle $\mathcal{A}$ des variables, puis un algorithme pour la résoudre. Notons $\delta_1, \delta_2, \ldots, \delta_l$ les valeurs distinctes des dates échues d_j des jobs $j \in J$ triées en ordre croissant. Le nombre l de dates échues distinctes peut être inférieur au nombre n de tâches lorsque certaines dates échues sont identiques. Par souci de simplicité, nous supposerons dans les formules et algorithmes de ce chapitre que $l = n$. En section 2.1 page 10, nous avons défini une affectation partielle $\mathcal{A}$ comme l'ensemble des domaines courants de toutes les variables ainsi que les notations propres aux réductions de domaine, par exemple une instanciation $x \leftarrow v$. Soit $K_{\mathcal{R}p}(\mathcal{A}) = \{k \in K \mid \max(D_k)\ \mathcal{R}\ \delta_p\}$ l'ensemble des fournées liées à la date échue δ_p par la relation arithmétique $\mathcal{R} \in \{<, \leq, =, \geq, >\}$. De la même manière, $J_{\mathcal{R}p}(\mathcal{A}) = \{j \in J \mid d_j\ \mathcal{R}\ \delta_p\}$ est l'ensemble des tâches liées à la date échue δ_p par la relation $\mathcal{R}$. Finalement, notons $P(\mathcal{A}, \tilde{K}) = \sum_{k \in \tilde{K}} \min(P_k)$ la durée totale minimale d'un ensemble de fournées $\tilde{K} \subseteq K$ pour l'affectation partielle $\mathcal{A}$.
Une instance $I(\mathcal{A})$ de la relaxation est composée de n blocs où chaque bloc $p \in J$ contient l'ensemble de fournées $K_{=p}(\mathcal{A})$. Chaque bloc p a une date échue δ_p et une durée d'exécution $\pi_p(\mathcal{A}) = P(\mathcal{A}, K_{=p}(\mathcal{A}))$. Étant donné que les blocs sont, par définition, numérotés en ordre croissant ($\delta_p < \delta_{p+1}$), le retard algébrique maximal de $I(\mathcal{A})$ est celui de la séquence de blocs ordonnancés dans cet ordre. Notons $C_p(\mathcal{A}) = \sum_{q=1}^{p} \pi_q(\mathcal{A})$ et $L_p(\mathcal{A}) = C_p(\mathcal{A}) - \delta_p$ la date d'achèvement et le retard algébrique du bloc p dans cette séquence. Le retard algébrique maximal d'une solution optimale de l'instance $I(\mathcal{A})$ est alors le suivant : $L(\mathcal{A}) = \max_J(L_p(\mathcal{A}))$.

La figure 8.1 illustre la construction et la résolution de la relaxation du problème $1|p\text{-}batch; b < n; non\text{-}identical|L_{max}$ à deux étapes différentes de la résolution. La capacité de la machine à traitement par fournées est $b = 10$. Le tableau 8.1(a) décrit les tâches à ordonnancer dans le problème original. Le tableau 8.1(b) donne les blocs de l'instance $I(\mathcal{A}_1)$ où $\mathcal{A}_1$ est une solution du problème, c'est-à-dire une affectation totale. Dans cet exemple, il y a moins de blocs ($l = 3$) que de tâches ($n = 4$). La figure 8.1(d) représente la solution optimale de la relaxation $I(\mathcal{A}_1)$ qui est également une solution optimale du problème original. La machine à traitement par fournées est représentée par un diagramme dans lequel les axes horizontaux et verticaux correspondent respectivement au temps et à la charge de la ressource. Une tâche est dessinée comme un rectangle dont la longueur et la hauteur représentent respectivement sa durée et sa taille. Les tâches dont les dates de début sont identiques appartiennent à la même fournée. La solution contient donc trois fournées : la première contient uniquement la tâche 1 ; la seconde contient les tâches 2 et 4 (sa charge est égale à $s_2 + s_4 = 9$, sa durée minimale est égale à $\max\{p_2, p_4\} = 9$ et sa date échue maximale est égale à $\min\{d_2, d_4\} = 2$) ; la troisième contient uniquement la tâche 3. Un bloc est représenté comme un rectangle dessiné en pointillés entourant les fournées lui appartenant. Le retard algébrique d'un bloc est représenté sur la droite de la figure par une ligne allant de sa date échue à sa date d'achèvement. Dans cet exemple, le bloc 1 contient les fournées 1 et 2 puisque $\max(D_1) = \max(D_2) = d_1$. Le bloc 2 est vide, car qu'il n'y a aucune fournée k telle que $\max(D_k) = \delta_2$. Le retard algébrique d'un bloc vide, dessiné comme une ligne en pointillés, est dominé par celui de son premier prédécesseur non vide. Le bloc 3 contient la fournée 3 constituée uniquement de la tâche 3.
Le tableau 8.1(c) donne les blocs de l'instance $I(\mathcal{A}_2)$ où $\mathcal{A}_2$ est une affectation partielle (la tâche 4 n'est pas encore placée). La figure 8.1(e) représente la solution optimale de l'instance $I(\mathcal{A}_2)$ dans laquelle la tâche 4 n'apparaît donc pas. Dans cet exemple, chaque bloc contient une seule fournée : le premier, le deuxième et le troisième bloc contiennent respectivement les fournées 1, 2 et 3. En fait, $\mathcal{A}_2$ est déduite

Job	1	2	3	4
s_j	8	7	5	2
p_j	5	8	7	9
d_j	2	7	10	2

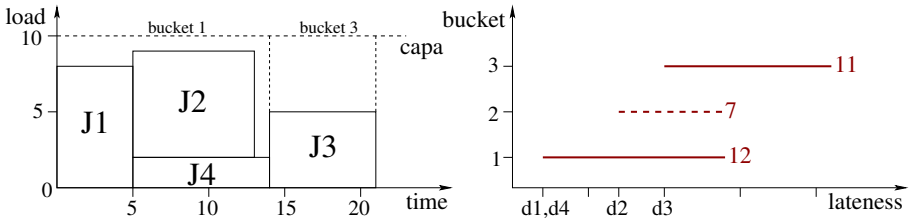
(a) Instance de $1||L_{max}$.

Bloc	1	2	3
δ_p	2	7	10
$\pi_p(\mathcal{A})$	14	0	7

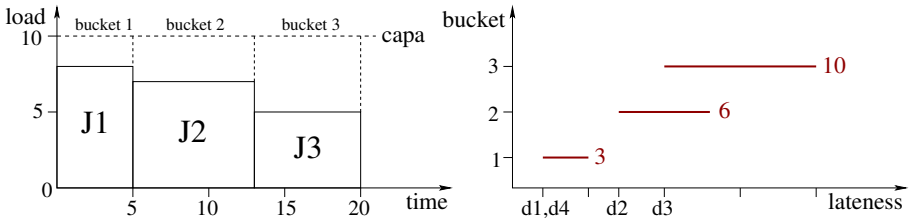
(b) Relaxation $I(\mathcal{A}_1)$.

Bloc	1	2	3
δ_p	2	7	10
$\pi_p(\mathcal{A})$	5	8	7

(c) Relaxation $I(\mathcal{A}_2)$.



(d) Solution de la relaxation associée à la solution (affectation totale) $\mathcal{A}_1$. $\mathcal{A}_1 = \{B_1 \leftarrow 1, B_2 \leftarrow 2, B_3 \leftarrow 3, B_4 \leftarrow 2\}$



(e) Solution de la relaxation associée à l'affectation partielle $\mathcal{A}_2$. La tâche 4 est absente, car elle n'est pas encore placée. $\mathcal{A}_2 = \{B_1 \leftarrow 1, B_2 \leftarrow 2, B_3 \leftarrow 3, B_4 \in [1, 4]\}$.

FIGURE 8.1 – Deux exemples de construction de la relaxation $I(\mathcal{A})$.

pendant la propagation à la racine de l'arbre de recherche par les contraintes (8.5) qui imposent que $B_1 = 1$, $B_2 = 2$, and $B_3 = 3$.

À chaque étape de la résolution, la construction de l'instance $I(\mathcal{A})$ peut être réalisée en $O(n)$. En effet, les dates échues sont triées en ordre croissant une fois pour toutes avant le début de la résolution avec une complexité en $O(n \log n)$. Ensuite, la construction de l'instance $I(\mathcal{A})$ consiste à affecter chaque fournée k à un bloc p. Ceci est réalisable en temps constant puisque le bloc p d'une fournée k est le bloc p vérifiant la condition $\delta_p = \max(D_k)$. L'insertion d'une fournée k dans son bloc et la mise à jour de la durée d'exécution du bloc sont aussi réalisées en temps constant. Par conséquent, la complexité totale de la construction de l'instance est $O(n)$, car le nombre de fournées m est inférieur au nombre de tâches n. Finalement, la résolution de l'instance $I(\mathcal{A})$ consiste simplement à séquencer les blocs en suivant leur numérotation, ce qui peut être réalisé en $O(n)$. Les sections suivantes présentent les règles de filtrages basées sur la relaxation du problème.

8.2.2 Filtrage de la variable objectif

Les règles de filtrage portant sur la variable objectif L_{max} reposent sur les deux propositions suivantes.

Proposition 1. *L est une fonction monotone d'un ensemble partiellement ordonné d'affectations vers les entiers naturels :*

$$\mathcal{A}' \subseteq \mathcal{A} \Rightarrow L(\mathcal{A}') \geq L(\mathcal{A}).$$

Démonstration. Les contraintes (8.1) imposent que la durée minimale d'une fournée dans $\mathcal{A}'$ soit supérieure à sa durée minimale dans $\mathcal{A}$, la durée totale d'un ensemble de fournées $\tilde{K}$ ne peut pas décroître lors de la restriction de $\mathcal{A}$ à $\mathcal{A}'$: $\forall \tilde{K} \subseteq K,\ P(\mathcal{A}, \tilde{K}) \leq P(\mathcal{A}', \tilde{K})$. De plus, puisque les contraintes (8.2) imposent que le placement d'une nouvelle tâche dans une fournée appartenant au bloc p implique son transfert vers un bloc p' tel que $p' \leq p$, l'ensemble des fournées appartenant au bloc p ou à un de ces

prédécesseurs ne peut pas diminuer lors de la restriction de $\mathcal{A}$ à $\mathcal{A}'$: $\forall p \in J,\ K_{\leq p}(\mathcal{A}) \subseteq K_{\leq p}(\mathcal{A}')$. Parallèlement, puisque P_k est une variable entière positive, $P(\mathcal{A}, \tilde{K})$ est une fonction croissante des sous-ensembles de fournées vers les entiers positifs : $\tilde{K} \subseteq \tilde{K}' \Rightarrow P(\mathcal{A}, \tilde{K}) \leq P(\mathcal{A}, \tilde{K}')$. Par conséquent, les inégalités suivantes sont valides :

$$\begin{aligned} C_p(\mathcal{A}) &= \sum_{1 \leq q \leq p} \pi_q(\mathcal{A}) = P(\mathcal{A}, K_{\leq p}(\mathcal{A})) \\ &\leq P(\mathcal{A}', K_{\leq p}(\mathcal{A})) \leq P(\mathcal{A}', K_{\leq p}(\mathcal{A}')) \\ &\leq C_p(\mathcal{A}'). \end{aligned}$$

Étant donné que la date d'achèvement d'un bloc p ne peut qu'augmenter lors de la restriction de $\mathcal{A}$ à $\mathcal{A}'$, la monotonie de la fonction $I(\mathcal{A})$ est prouvée de la manière suivante :

$$(8.1) \wedge (8.2) \quad \Rightarrow \quad \forall p \in J,\ C_p(\mathcal{A}') \geq C_p(\mathcal{A}) \quad \Rightarrow \quad L(\mathcal{A}) \leq L(\mathcal{A}')$$

□

Proposition 2. *Lorsque toutes les durées d'exécution et toutes les dates échues des fournées sont fixées, un ordonnancement optimal de la relaxation $I(\mathcal{A})$ correspond aussi à un ordonnancement optimal pour toute solution réalisable obtenue à partir de $\mathcal{A}$.*

Démonstration. Lorsque toutes les durées d'exécution et dates échues des fournées sont fixées, les durées et retards des blocs ne seront plus modifiés jusqu'à ce qu'une solution réalisable soit découverte. Par conséquent, la proposition est vraie si le retard algébrique maximal de la relaxation $L(\mathcal{A})$ est égal au retard algébrique maximal de toute solution obtenue à partir de $\mathcal{A}$. Comme toutes les fournées d'un bloc p ont la même date échue δ_p, toutes les séquences de ses fournées sont équivalentes pour la règle EDD. Ainsi, un ordonnancement optimal des blocs dans la relaxation correspond à une classe d'ordonnancement optimale des fournées (respectant la règle EDD) pour le problème initial. Remarquez qu'il est souvent préférable d'ordonnancer les fournées d'un bloc par ordre croissant de durée d'exécution puisque cela diminue le retard moyen des fournées. □

Grâce à la proposition 2, nous déduisons la règle de filtrage du retard final (*final lateness filtering rule* – FF) qui résout la relaxation pour instancier la variable objectif dès que toutes les durées et toutes les dates échues des fournées sont instanciées :

$$\forall k \in K, P_k \text{ et } D_k \text{ sont instanciées } \Rightarrow L_{max} \leftarrow L(\mathcal{A}). \tag{FF}$$

Remarquez que cette règle peut se déclencher avant le placement de toutes les tâches, car la propagation peut réduire les domaines à un singleton. Un corollaire des propositions 1 et 2 est qu'à chaque étape de la résolution, le retard algébrique maximal $L(\mathcal{A})$ de l'instance de la relaxation $I(\mathcal{A})$ est une borne inférieure du retard algébrique maximal de tout ordonnancement restreignant $\mathcal{A}$. Par conséquent, la valeur minimale de l'objectif peut être mise à jour après chaque modification pertinente des domaines par la règle de filtrage du retard (*lateness filtering rule* – LF) :

$$\exists k \in K, \min(P_k) \text{ ou } \max(D_k) \text{ ont changé} \Rightarrow \min(L_{max}) \leftarrow L(\mathcal{A}) \tag{LF}$$

8.2.3 Filtrage basé sur le coût du placement d'une tâche

La règle de filtrage basée sur le coût du placement d'une tâche (*cost-based domain filtering rule of assignments* – AF) réduit l'espace de recherche en se basant sur le coût marginal associé au placement d'une tâche j dans une fournée k ($B_j \leftarrow k$). L'idée est d'éliminer, après chaque modification pertinente des domaines, chaque tâche j comme candidate pour un placement dans la fournée k, si le coût marginal associé à son placement dans la fournée k dépasse la meilleure borne supérieure découverte jusqu'à présent, ou plus formellement :

$$\begin{aligned} &\exists k \in K, \min(P_k) \text{ ou } \max(D_k) \text{ ont changé} \Rightarrow \\ &\qquad \forall j \in J, \text{ tel que } |\mathcal{D}(B_j)| > 1 \text{ et } \forall k \in \mathcal{D}(B_j), \\ &\qquad\qquad L(\mathcal{A} \cap \{B_j \leftarrow k\}) > \max(L_{max}) \quad \Rightarrow \quad B_j \not\leftarrow k. \end{aligned} \tag{AF}$$

$\mathcal{A} \cap \{B_j \leftarrow k\}$ est une abréviation pour $\mathcal{A} \cap \{B_j \leftarrow k, \min(P_k) \leftarrow p_j, \max(D_k) \leftarrow d_j\}$. En effet, la propagation des contraintes (8.1) et (8.2) après le placement $B_j \leftarrow k$ implique que $\min(P_k) \leftarrow p_j$ et $\max(D_k) \leftarrow d_j$.
Un algorithme de filtrage basique peut calculer le coût marginal de chaque placement réalisable avec une complexité $O(n^3)$. Dans la section 8.5, nous proposons une version de cet algorithme de complexité $O(nm)$ basée sur un calcul incrémental des coûts marginaux.

8.2.4 Filtrage basé sur le coût du nombre de fournées non vides

Dans cette section, nous introduisons une règle de filtrage basée sur le coût marginal associé au nombre de fournées non vides M. Notons $K^\star = \{k \in K \mid \exists j \in J,\ B_j = k\}$ l'ensemble des fournées non vides. Nous supposons que $|K^\star| = \max\{k \in K^\star\}$, c'est-à-dire les tâches sont placées dans les premières fournées. Notons $J^\star = \{j \in J \mid \mathcal{D}(B_j) > 1 \wedge \max(B_j) > |K^\star|\}$ l'ensemble des tâches *disponibles*, c'est-à-dire les tâches qui peuvent être placées dans une nouvelle fournée. La règle (PF1) actualise le nombre possible de fournées non vides en considérant leur nombre actuel et le nombre de tâches disponibles.

$$\min(M) \leftarrow |K^\star| \qquad \max(M) \leftarrow |K^\star| + |J^\star| \tag{PF1}$$

Cette règle (appliquée aussi par `pack`) est nécessaire pour garantir la validité des raisonnements présentés ci-dessous. Soit $\mathcal{A}' = \mathcal{A} \cap \left(\cup_{j \in \tilde{J}} \{B_j \leftarrow k_j\}\right)$ une affectation partielle restreinte par le placement d'un sous-ensemble $\tilde{J} \subseteq J^\star$ de tâches disponibles à de nouvelles fournées, c'est-à-dire un ensemble de fournées vides avec des indices distincts ($\forall j \in \tilde{J},\ k_j > |K^\star|$ et $\forall i \neq j \in \tilde{J},\ k_i \neq k_j$). Par conséquent, la date d'achèvement et le retard algébrique de chaque bloc p doivent être actualisés en fonction de l'augmentation de la durée de ces prédécesseurs. En effet, la nouvelle date d'achèvement d'un bloc p est égale à sa date d'achèvement avant les placements à laquelle s'ajoute l'augmentation de sa durée et celles de ses prédécesseurs : $C_p(\mathcal{A}') = C_p(\mathcal{A}) + \sum_{j \in \tilde{J}_{\leq p}} p_j$. Malheureusement, la combinatoire des placements possibles augmente exponentiellement ce qui rend ce type de filtrage difficile et coûteux.
Pour pallier cet inconvénient, nous calculons une borne inférieure de $C_p(\mathcal{A}')$ en calculant une borne inférieure sur la date d'achèvement $C_p(\mathcal{A} \cap \{M \leftarrow q\})$ d'un bloc p pour un nombre total q de fournées non vides de la manière suivante. Notons $\Pi(q)$ la somme des q plus petites durées parmi celles des tâches disponibles. Si on crée $q - |K^\star|$ nouvelles fournées avec des tâches disponibles, il y a au moins $q - |K^\star| - |J^\star_{>p}|$ nouvelles fournées ordonnancées dans ou avant le bloc p. Dans ce cas, la date d'achèvement d'un bloc p après la création de $q - |K^\star|$ fournées est supérieure à sa date d'achèvement avant la création des nouvelles fournées additionnée à la somme minimale $\Pi(q - |K^\star| - |J^\star_{>p}|)$ des durées d'exécution de $q - |K^\star| - |J^\star_{>p}|$ tâches disponibles :

$$\begin{aligned} C_p(\mathcal{A} \cap \{M \leftarrow q\}) &= C_p(\mathcal{A}) + \Pi(q - |K^\star| - |J^\star_{>p}|) \\ &\leq C_p\left(\mathcal{A} \cap \left(\cup_{j \in \tilde{J}} \{B_j \leftarrow k_j\}\right)\right) \qquad \forall \tilde{J} \subseteq J^\star,\ |\tilde{J}| = q - |K^\star|. \end{aligned}$$

Ainsi, le retard algébrique maximal d'une solution obtenue à partir de $\mathcal{A}$ contenant exactement q fournées non vides est supérieur à $L(\mathcal{A} \cap \{M \leftarrow q\})$. La règle (PF2) actualise la borne inférieure de l'objectif en fonction du coût marginal associé au nombre minimal de fournées non vides. La règle (PF3) décrémente le nombre maximal de fournées non vides tant que son coût marginal dépasse la meilleure borne supérieure découverte jusqu'à présent.

$$\min(L_{max}) \leftarrow L(\mathcal{A} \cap \{M \leftarrow \min(M)\}) \tag{PF2}$$
$$L(\mathcal{A} \cap \{M \leftarrow \max(M)\}) > \max(L_{max}) \Rightarrow \max(M) \leftarrow \max(M) - 1 \tag{PF3}$$

Après chaque modification pertinente des domaines, la règle de filtrage basée sur le coût du nombre de fournées non vides (*cost-based domain filtering rule based on bin packing* – PF) applique les trois règles de filtrage décrites ci-dessus.

$$(\exists j \in J, \mathcal{D}(B_j) \text{ a changé}) \vee (\exists k \in K, \min(P_k) \text{ ou } \max(D_k) \text{ ont changé}) \Rightarrow$$
$$\text{Apply rules (PF1), (PF2) and (PF3)} \tag{PF}$$

Le calcul de la fonction Π est réalisé en $O(n \log n)$ pendant l'initialisation en triant et sommant les durées d'exécution des tâches disponibles. Les règles (PF1) et (PF2) sont appliquées en temps linéaire, puisque la construction et la résolution de $I(\mathcal{A})$ a une complexité en $O(n)$. La règle (PF3) est appliquée jusqu'à ce que le coût marginal associé au nombre maximal de fournées non vides devienne satisfiable, c'est-à-dire au plus $|J^\star|$ fois. Par conséquent, la complexité du filtrage basé sur le coût du nombre de fournées non vides est $O(n^2)$.

La figure 8.2 illustre le calcul du coût marginal associé à la présence d'exactement $q = 4$ fournées non vides pour l'instance introduite dans le tableau 8.1(a) et une affectation vide $\mathcal{A}_3$ (avant la propagation au nœud racine). L'ensemble des tâches disponibles $J^\star$ est l'ensemble des tâches J et toutes les fournées sont vides ($K^\star = \emptyset$). Par conséquent, les règles (PF1) et (PF2) ne modifient pas le domaine de M. Puisque la date d'achèvement $C_p(\mathcal{A}_3)$ de chaque bloc p est nul, sa nouvelle date d'achèvement $C_p(\mathcal{A}_3 \cap \{M \leftarrow 4\})$ est égale à $\Pi(q - |J^\star_{>p}|)$. Ainsi, le bloc 1 s'achève à l'instant $\Pi(4-|\{2,3\}|) = p_1+p_3 = 12$, le bloc 2 s'achève à l'instant $\Pi(4-|\{3\}|) = p_1+p_3+p_2 = 20$, et le bloc 3 s'achève à l'instant $\Pi(4-|\emptyset|) = p_1+p_3+p_2+p_4 = 29$. En supposant que la meilleure borne supérieure découverte pendant la résolution soit égale à 15, alors le nombre maximal de fournées non vides est réduit de 4 à 3. On doit alors appliquer à nouveau la règle (PF2) pour $q = 3$.

8.3 Stratégie de branchement

À chaque nœud de l'arbre de recherche, la stratégie de branchement choisit la variable B_j associée à une tâche j en utilisant une heuristique de sélection de variable, et place la tâche j dans la fournée k choisie par une heuristique de sélection de valeur. Lors d'un retour arrière, le branchement interdit le placement de la tâche sélectionnée dans la fournée sélectionnée. Si une fournée redevient vide, une règle d'élimination de symétrie est appliquée pour éliminer les fournées équivalentes, c'est-à-dire les autres fournées vides, comme candidates pour le placement de la tâche courante. Remarquez que cette règle dynamique d'élimination de symétrie généralise l'idée qui est à la base de l'introduction des contraintes (8.5), parce qu'elle évite de placer les j_0 plus grandes tâches dans une autre fournée lors d'un retour arrière.
Plusieurs heuristiques de sélection de variable exploitant les informations sur les durées, tailles et dates échues peuvent être utilisées. Nous avons choisi une heuristique classique appelée *complete decreasing* [67] qui place les tâches par ordre décroissant de taille. En effet, des expérimentations préliminaires ont montré que privilégier le placement des grandes tâches améliore le filtrage des contraintes `pack` et `sequenceEDD` par rapport aux variantes se basant aussi sur les durées d'exécution et les dates échues.
Nous avons sélectionné deux heuristiques de sélection de valeur classiques pour les problèmes de placement à une dimension : *first fit* qui choisit la première fournée possible pour une tâche ; et *best fit* qui choisit la fournée la plus chargée, c'est-à-dire avec le moins d'espace restant. Par ailleurs, nous proposons une nouvelle heuristique de sélection de valeurs appelée *batch fit* qui choisit la fournée ayant la plus petite valeur $\gamma(B_j \leftarrow k)$ où $\gamma(B_j \leftarrow k)$ est une estimation de la compatibilité du placement dans l'instance $I(\mathcal{A})$ calculée de la manière suivante :

$$\gamma(B_j \leftarrow k) = \frac{|\min(P_k) - p_j|}{\max_J(p_j) - \min_J(p_j)} + \frac{|\max(D_k) - d_j|}{\max_J(d_j) - \min_J(d_j)}$$

L'idée sur laquelle repose cette heuristique est qu'une solution « parfaite » est composée de fournées contenant des tâches dont les durées et dates échues sont toutes identiques.

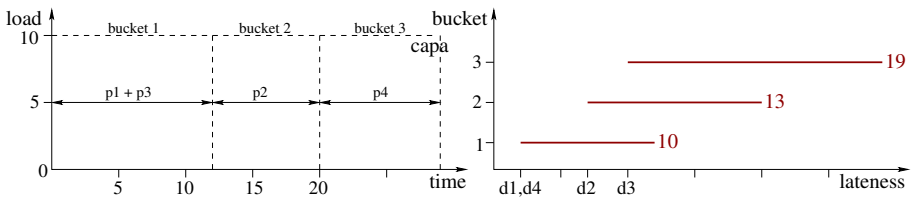


FIGURE 8.2 – Exemple de calcul de $L(\mathcal{A}_3 \cap \{M \leftarrow 4\})$ pour une affectation (vide) $\mathcal{A}_3$.
$\mathcal{A}_3 = \{B_1 \in [1,4], B_2 \in [1,4], B_3 \in [1,4], B_4 \in [1,4]\}$.

8.4 Expérimentations

Cette section récapitule les expérimentations menées pour évaluer notre approche sur le jeu d'instances présenté en section 5.3. Lors de la résolution d'une instance, nous utilisons la procédure d'optimisation top-down décrite en section 2.4, car nous n'avons aucune garantie sur la borne inférieure initiale. Les sections 8.4.1 et 8.4.2 évaluent respectivement les performances des règles de filtrage et heuristiques de sélection de valeur. Finalement, les résultats de notre approche sont comparés à ceux d'une formulation en programmation mathématique en section 8.4.3, puis à une approche par branch-and-price en section 8.4.4. Toutes les expérimentations ont été réalisées sur une grille de calcul composée de machines tournant sous Linux où chaque nœud possède 48 GB de mémoire vive et deux processeurs à quadruple cœur cadencé s à 2.4 GHz. Notre implémentation est basée sur le solveur `choco` (voir chapitre 9) pour lequel la contrainte `pack` est disponible depuis la livraison 2.0.0. Un module externe fournit une implémentation de la contrainte `sequenceEDD` et de notre nouvelle heuristique de sélection de valeur. Ce module utilise l'algorithme pour la règle de filtrage (AF) présentée en section 8.5. Une limite de temps de 3600 secondes (1h) est fixée pour la résolution.

8.4.1 Performance des règles de filtrage

L'amélioration du filtrage d'une contrainte est souvent bénéfique à la résolution, mais il arrive quelquefois que les performances d'un algorithme simple surpassent celles d'algorithmes plus compliqués. Par conséquent, les sections 8.4.1.1 et 8.4.1.2 évaluent respectivement l'efficacité du filtrage et son utilité pendant la résolution (voir section 8.4.1.2). Dans cette section, le nom d'une règle de filtrage est une abréviation qui représente leur structure hiérarchique, c'est-à-dire (FF) $\subset$ (LF) $\subset$ (AF) $\subset$ (PF). Par exemple, l'abréviation (PF) signifie que toutes les règles de filtrage sont appliquées.

8.4.1.1 Calcul d'une borne inférieure destructive

Nous comparons la puissance des règles de filtrage en calculant des bornes inférieures destructives (procédure destructive-lower-bound de la section 2.4). Les trois premières bornes inférieures destructives sont calculées en utilisant les règles de filtrage (LF), (AF), et (PF). Remarquez que nous ignorons délibérément la règle (FF), car il est peu probable qu'elle réduise des domaines dans ce contexte. Ces trois premières bornes étant calculées très rapidement, trois autres (meilleures) bornes sont calculées en intégrant une étape de *shaving* comme suggéré par Rossi *et al.* [10]. Le *shaving* est semblable à une preuve par contradiction. On propage le placement de la tâche j dans la fournée k. Si une insatisfiabilité est détectée, alors le placement n'est pas valide et on peut supprimer la tâche j des candidates pour un placement dans la fournée k. Pour limiter le temps de calcul, *shaving* est utilisé une seule fois pour chaque placement $B_j \leftarrow k$.

Notons ub la meilleure borne supérieure découverte pour une instance donnée [1]. La qualité d'une borne inférieure lb pour une instance donnée est estimée grâce à la formule $(100 \times (lb + d_{max})) \div (ub + d_{max})$ (inspiré des travaux de Malve et Uzsoy [135]). Cette mesure est égale à l'écart à l'optimum pour les instances contenant moins de 20 tâches, car tous les optimums de ces instances ont été prouvés. Par ailleurs, cette mesure est très proche de l'écart à l'optimum pour les instances contenant 50 tâches, car seules deux instances ne sont pas résolues optimalement. Le tableau 8.1 donne la qualité moyenne des deux types de borne inférieure destructive (colonnes 5–7 et 8–10) en fonction du nombre de tâches des instances. L'efficacité de ces bornes est aussi comparée à celle des bornes inférieures obtenues après la propagation initiale (colonnes 2–4). Les temps de calcul sont mentionnés uniquement pour les bornes inférieures destructives renforcées par du *shaving* (colonnes 11–13), parce qu'ils sont négligeables pour les autres bornes ($\ll 1$ second). Remarquez que la présence des contraintes (8.5) contribue grandement à améliorer la qualité de ces bornes inférieures. La règle (AF) n'améliore par la borne inférieure durant la propagation initiale, alors qu'elle améliore significativement toutes les bornes inférieures destructives. Dans tous les cas, la règle (PF) améliore légèrement la qualité des bornes obtenues avec la règle (AF). Par ailleurs, la qualité les bornes inférieures destructives sans *shaving* est satisfaisante tout en conservant un temps de calcul négligeable. Par contre, la phase de *shaving* améliore ces bornes, mais entraîne une

1. `http://www.mines-nantes.fr/en/Sites-persos/Christelle-GUERET/Batch-processing-problems`

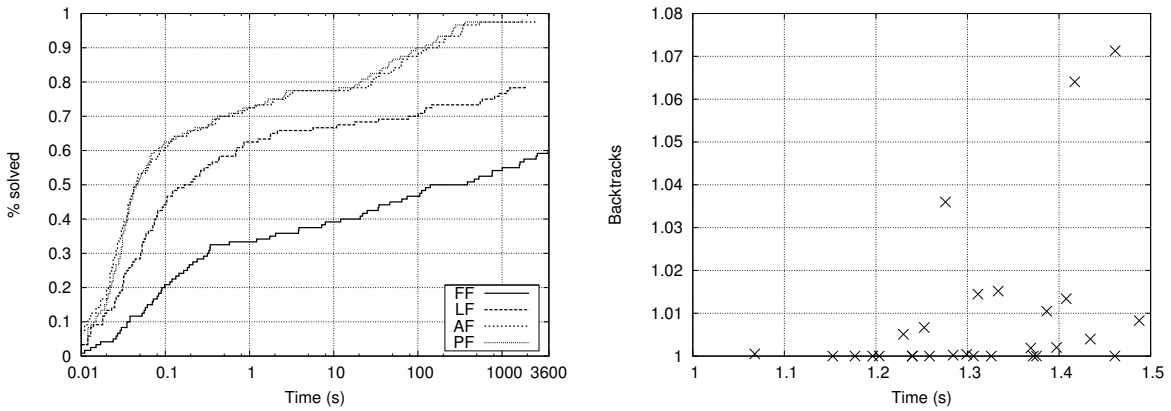
	Initial Propag.			Destr. LB			Destr. LB + Shaving					
n	LF	AF	PF	LF	AF	PF	LF	AF	PF	$\bar{t}_{\mathrm{LF}}$	$\bar{t}_{\mathrm{AF}}$	$\bar{t}_{\mathrm{PF}}$
10	89.4	89.4	89.7	89.4	97.9	98.1	93.4	99.0	**99.2**	0.07	**0.05**	**0.05**
20	90.1	90.1	90.4	90.1	95.6	95.7	93.6	96.8	**97.0**	0.17	0.15	**0.1**
50	89.7	89.7	90.2	89.7	93.6	93.6	91.8	**94.1**	**94.1**	2.71	2.71	**1.43**
75	88.5	88.5	89.1	88.5	91.4	91.5	89.8	91.7	**91.8**	8.8	12.01	**5.25**
100	87.2	87.2	87.6	87.2	89.4	89.4	88.3	89.6	**89.7**	23.20	29.48	**14.04**

TABLE 8.1 – Qualité des bornes inférieures destructives.

augmentation importante du temps de calcul. Remarquez que la règle (PF) améliore la qualité des bornes inférieures destructives avec *shaving* tout en diminuant leurs temps de calcul de moitié.

8.4.1.2 Complete decreasing first fit

La procédure *complete decreasing first fit* [67] place les tâches par ordre décroissant de taille en plaçant chaque tâche dans la première fournée pouvant la contenir. Nous désactivons le mécanisme de calcul des bornes inférieures destructives pour réduire l'influence des règles de filtrage sur la stratégie de branchement. Nous avons sélectionné un sous-ensemble d'instances qui peuvent être résolues optimalement par la plupart des règles de filtrage, depuis (FF) jusqu'à (PF), pour que les comparaisons puissent être réalisées dans un laps de temps raisonnable. Dans ce but, nous avons sélectionné toutes les instances avec moins de 50 tâches (120 instances).
La figure 8.3(a) donne le pourcentage d'instances résolues en fonction du temps (en secondes) pour chacune des règles de filtrage. Tout d'abord, les performances de la règle (FF) montrent que notre modèle est capable de capturer des solutions optimales même lorsque le filtrage de la variable objectif n'a lieu qu'après l'instanciation de toutes les durées et dates échues des fournées. Nous pensons que notre modèle en programmation par contraintes est très efficace pour détecter les placements des jobs menant à des ordonnancements équivalents par rapport à la règle EDD. Ensuite, la règle (LF) basée uniquement sur la relaxation du problème améliore la qualité des solutions et les temps de calcul par rapport à la règle (FF), même si un certain nombre d'instances ne sont pas résolues optimalement. Finalement, seules deux instances avec 50 tâches ne sont pas résolues optimalement dans la limite de temps quand on applique la règle (AF) ou (PF). Par contre, la figure ne montre pas clairement le gain offert par la règle (PF) par rapport à la règle (AF).
Pour répondre à cette question, la figure 8.3(b) compare en détail la résolution des instances avec les règles (AF) et (PF). Chaque point représente une instance dont la coordonnée x est le quotient du temps



(a) Comparaison du temps de résolution ($n \leq 50$). (b) (AF) ÷ (PF) ($n = 50$).

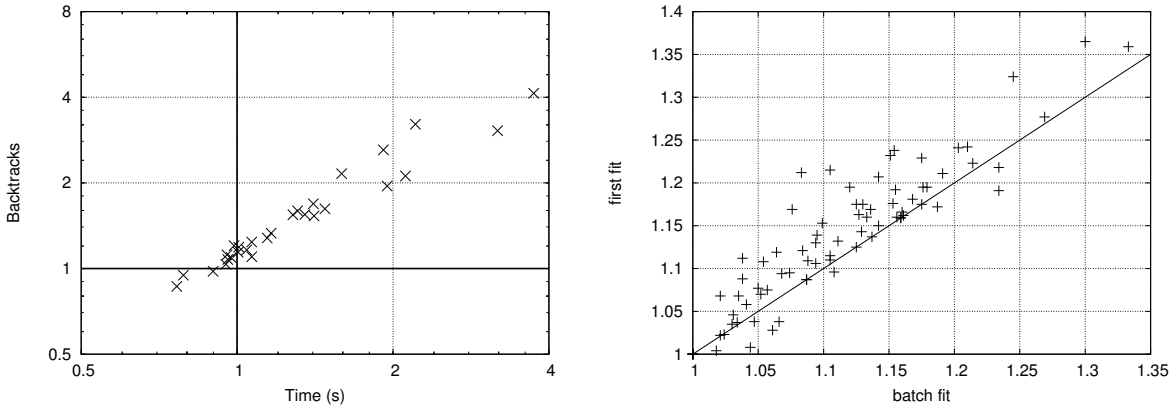
FIGURE 8.3 – Comparaison des règles de filtrage en utilisant *complete decreasing first fit*.

de résolution avec la règle (AF) sur celui avec (PF) et la coordonnée y est le quotient du nombre de backtracks avec la règle (AF) sur celui avec (PF). Nous ne considérons que les instances dont le temps de résolution est supérieur à 2 secondes, ce qui ne se produit que pour celles avec 50 tâches. Tous les points sont situés en haut à droite du point (1,1), car l'utilisation de la règle (PF) améliore la résolution de toutes les instances sans exception. En fait, la règle (PF) a un comportement similaire à celui observé pendant le calcul des bornes inférieures destructives. Son utilisation diminue toujours le temps de résolution et certaines instances sont même résolues approximativement 30 % plus vite, mais le nombre de backtracks est presque identique avec les deux règles.

8.4.2 Performance des heuristiques de sélection de valeur

Nous mesurons maintenant l'influence de l'heuristique de sélection de valeur sur la résolution. Dans ce but, nous résolvons toutes les instances en utilisant toutes les règles de filtrage, mais aucune borne inférieure destructive. Nous comparerons seulement *batch fit* à *first fit*, parce que *first fit* et *best fit* obtiennent des performances équivalentes dans le cadre de notre approche. En effet, la différence entre les temps de résolution avec *first fit* et *best fit* est toujours inférieure à 5 secondes pour les instances avec 50 tâches et cette différence correspond au plus à 2 % du temps de résolution. De plus, les deux heuristiques atteignent toujours la même borne supérieure pour les instances avec plus de 50 tâches.
La figure 8.4(a) étudie les effets de l'heuristique de sélection de valeur pour les instances avec 50 tâches et un temps de résolution supérieur à 2 secondes. Chaque point représente une instance dont la coordonnée x est le quotient du temps de résolution avec *first fit* sur celui avec *batch fit* et la coordonnée y est le quotient du nombre de backtracks avec *first fit* sur celui avec *batch fit*. Les deux axes ont des échelles logarithmiques. Tous les points situés au-dessus ou à droite du point (1,1) représentent des instances dont la résolution est améliorée par l'utilisation de *batch fit* (quadrant en haut à droite). Au contraire, les quelques points situés en dessous ou à gauche du point (1,1) sont des instances pour lesquelles il est préférable de garder *first fit* (quadrant en bas à gauche). Comme espéré, tous les points se situent autour de la diagonale ($x = y$) puisque le nombre de backtracks est à peu près proportionnel au temps de résolution (les quadrants en haut à gauche et en bas à droite sont vides).
La figure 8.4(b) analyse les effets de l'heuristique de sélection de valeur sur les instances avec plus de 50 tâches. Notons lb la meilleure borne supérieure connue pour une instance [2]. La qualité d'une solution ub est estimée en mesurant son écart à lb grâce à une formule inspirée par les travaux de Malve et Uzsoy [135] : $(ub + d_{max}) \div (lb + d_{max})$. Cet écart est généralement supérieur à l'écart à l'optimum puisque ces instances n'ont pas toujours été résolues optimalement. Chaque point représente une instance dont

2. `http://www.mines-nantes.fr/en/Sites-persos/Christelle-GUERET/Batch-processing-problems`



(a) *first fit* ÷ *batch fit* ($n = 50$).

(b) Comparaison des écart à la borne inférieure ($n > 50$).

FIGURE 8.4 – Comparaison des heuristiques de sélection de valeur.

la coordonnée x est l'écart à la borne inférieure obtenu avec *batch fit* alors que la coordonnée y est celui obtenu avec *first fit*. Tous les points situés au-dessus de la droite $(x = y)$ correspondent à des instances pour lesquelles *batch fit* a retourné une meilleure solution. L'utilisation de *batch fit* améliore globalement la résolution même si les performances sur quelques rares instances sont dégradées.

8.4.3 Comparaison avec un modèle en programmation mathématique

Notre approche est maintenant comparée à une formulation en programmation mathématique inspirée par les travaux de Daste *et al.* [133]. Soit x_{jk} une variable booléenne prenant la valeur 1 si la tâche j est placée dans la k-ème fournée de la séquence. Les variables entières positives P_k, D_k, et C_k représentent respectivement la durée d'exécution, la date échue et la date d'achèvement de la k-ème fournée.

$$\min L_{max} \tag{8.6}$$

Subject to :

$$\sum_{k \in K} x_{jk} = 1 \qquad \forall j \in J \tag{8.7}$$

$$\sum_{j \in J} s_j x_{jk} \leq b \qquad \forall k \in K \tag{8.8}$$

$$p_j x_{jk} \leq P_k \qquad \forall j \in J,\ \forall k \in K \tag{8.9}$$

$$C_{k-1} + P_k = C_k \qquad \forall j \in J,\ \forall k \in K \tag{8.10}$$

$$(d_{max} - d_j)(1 - x_{jk}) + d_j \geq D_k \qquad \forall j \in J,\ \forall k \in K \tag{8.11}$$

$$D_{k-1} \leq D_k \qquad \forall k \in K \tag{8.12}$$

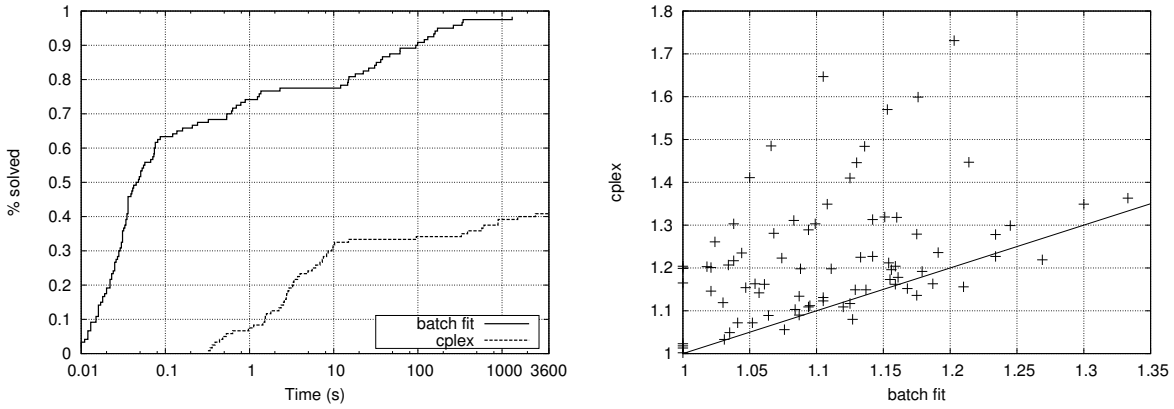
$$C_k - D_k \leq L_{max} \qquad \forall k \in K \tag{8.13}$$

$$\forall j \in J,\ \forall k \in K,\ x_{jk} \in \{0, 1\}$$

$$\forall k \in K,\ C_k \geq 0,\ P_k \geq 0,\ D_k \geq 0$$

Les contraintes (8.6) définissent le critère d'optimalité qui consiste à minimiser le retard algébrique maximal des tâches. Les contraintes (8.7) imposent que chaque tâche soit placée dans exactement une fournée et les contraintes (8.8) imposent que la capacité b de la machine soit respectée dans chaque fournée. Ces contraintes définissent le modèle pour le problème de *bin packing* proposé par Martello et Toth [64]. Les contraintes (8.9) imposent que la durée d'exécution de chaque fournée k soit égale à la durée d'exécution maximale de ses tâches. Les contraintes (8.10) imposent que la date d'achèvement de la k-ème fournée soit égale à la date d'achèvement de la $(k-1)$-ème fournée additionnée à la durée d'exécution de la k-ème fournée, c'est-à-dire qu'une solution est une séquence de fournée sans temps mort en suivant leur numérotation. Il est nécessaire d'ajouter une fournée fictive terminant à l'origine du temps ($C_0 = 0$). Les contraintes (8.11) imposent que la date échue D_k de la k-ème fournée soit égale à la plus petite date échue de ses tâches. Les contraintes (8.12) imposent le respect de la règle EDD par la séquence de fournées. Ces contraintes améliorent la résolution, car les permutations des fournées menant à un ordonnancement équivalent en appliquant la règle EDD sont interdites, mais elles ne sont pas nécessaires à la correction du modèle. Les contraintes (8.13) sont la définition du retard des fournées et imposent que la valeur de l'objectif soit égale au retard algébrique maximal. Finalement, une solution de la formulation en programmation mathématique est un placement réalisable des tâches dans une séquence de fournées respectant la règle EDD. À l'opposé du modèle en programmation par contraintes, cet encodage de la solution ne détecte pas toutes les permutations équivalentes des fournées.

Notre approche applique l'heuristique de sélection de valeur *batch fit* et toutes les règles de filtrage. Les comparaisons ont été réalisées avec la formulation en programmation mathématique implémentée comme un modèle Ilog OPL 6.1.1 et résolue par `Ilog Cplex` 11.2.1. Le solveur `Ilog Cplex` fournit un outil de configuration automatique qui tente de trouver la meilleure combinaison de ses paramètres. Cet outil a été seulement appliqué sur les instances avec 10 tâches, car il est nécessaire de les résoudre optimalement plusieurs fois. Tout d'abord, cet outil a modifié le type de *coupe* pour réduire le nombre de branches explorées. Une *coupe* est une contrainte ajoutée au modèle pour supprimer des solutions optimales fractionnaires de la relaxation continue. Ensuite, cet outil privilégie la satisfiabilité en plaçant la tâche j dans la fournée k dans la première branche ($x_{jk} = 1$) à la place du comportement par défaut qui interdit ce placement ($x_{jk} = 0$).
La figure 8.5(a) montre le pourcentage d'instances résolues optimalement en moins de 3600 secondes (1h)



(a) Comparaison du temps de résolution ($n \leq 50$). (b) Comparaison des écart à la borne inférieure ($n > 50$).

FIGURE 8.5 – Comparaison à la formulation mathématique.

comme une fonction du temps de résolution Notre approche en programmation par contraintes obtient de meilleurs résultats que la formulation mathématique : elle résout un plus grand nombre d'instances avec des temps de résolution inférieurs d'un ordre de grandeur.
La figure 8.5(b) compare les écarts à la meilleure borne inférieure connue pour les instances avec plus de 50 tâches. La limite de temps de `Ilog Cplex` est relevée à 43200 secondes (12h). Chaque point représente une instance dont la coordonnée x est l'écart à la borne inférieure obtenu par notre approche et la coordonnée y est l'écart obtenu par `Ilog Cplex`. Tous les points situés au-dessus de la droite ($x = y$) sont des instances pour lesquelles notre approche a retourné la meilleure solution. Malgré une limite de temps d'une heure, notre approche renvoie une meilleure solution que la formulation mathématique pour la grande majorité des instances. De plus, la différence entre les deux solutions est très faible lorsque la formulation mathématique retourne la meilleure solution, alors qu'elle peut être significative lorsqu'il s'agit de notre approche.

8.4.4 Comparaison avec une approche par branch-and-price

Dans cette section, nous nous comparons à l'approche par branch-and-price de Daste *et al.* [151] décrite en section 5.3. Nous rappelons qu'une solution du problème maître est une séquence réalisable de fournées et que chaque colonne représente une fournée. À chaque itération, on résout un sous-problème pour déterminer une colonne (fournée) qui améliore la solution du problème maître par un algorithme glouton, puis si nécessaire, par une méthode exacte d'énumération. Leur approche a été testée sur le jeu d'instances utilisé présentement avec une limite de temps de 3600 secondes (1h), mais en se limitant aux instances avec moins de 50 tâches. Toutes leurs expérimentations ont été réalisées sur un Pentium avec un processeur cadencé à 2.66 GHz et 1 GB de mémoire vive. Les résultats détaillés sur ces instances ne sont pas précisés. Leur approche par branch-and-price résout respectivement 55% et 8% des instances contenant 20 et 50 tâches. Leurs expérimentations montrent que l'approche par branch-and-price est plus efficace que la formulation mathématique, mais celle-ci reste moins performante que notre approche en programmation par contraintes qui résout toutes les instances contenant 20 tâches en moins d'une seconde et 95% des instances contenant 50 tâches avec un temps moyen de résolution inférieur à 100 secondes.

8.5 Conclusion

Dans ce chapitre, nous avons présenté une approche par programmation par contraintes pour minimiser le retard algébrique maximal d'une machine à traitement par fournées sur laquelle un nombre fini de tâches avec des tailles différentes doivent être ordonnancées. Cette approche exploite une nouvelle

contrainte globale pour l'optimisation basée sur une relaxation du problème dont on déduit des règles de filtrage basées sur les coûts. L'exploration de l'arbre de recherche est améliorée grâce à des heuristiques de sélection de variable et de valeur. Les résultats expérimentaux montrent d'abord l'intérêt des différentes composantes de notre approche. Les comparaisons avec une formulation mathématique et une approche par branch-and-price révèlent que notre approche fournit de meilleures solutions avec des temps de résolution inférieurs d'un ordre de grandeur.
Dans de futurs travaux, nous souhaitons adapter cette approche pour d'autres critères d'optimalité relatifs aux dates d'achèvement (C_{max}, $\sum C_j$, $\sum w_j C_j$). D'autres sujets de recherche concernent le cas à plusieurs machines à traitement par fournées ou la présence de contraintes additionnelles, notamment des dates de disponibilité qui sont actuellement incompatibles avec notre approche.

Annexe 8.A : un algorithme quadratique pour le filtrage du placement des tâches

Nous rappelons qu'un algorithme basique peut calculer indépendamment les coûts marginaux de tous les placements possibles avec une complexité $O(n^3)$. Dans cette section, nous décrivons un algorithme de complexité $O(nm)$ basé sur le calcul incrémental des coûts marginaux nommé jobCBF. L'idée principale est d'exploiter la formule (8.14) pour calculer rapidement les coûts marginaux en distinguant plusieurs cas de placement. Par exemple, tous les placements d'une tâche dans une fournée vide ont le même coût marginal. En fait, la date d'achèvement d'un bloc p après le placement de la tâche j dans la fournée $k \in \mathcal{D}(B_j)$ est la suivante :

$$C_p(\mathcal{A} \cup \{B_j \leftarrow k\}) = \begin{cases} C_p(\mathcal{A}) & \text{if } \delta_p < \min(d_j, \max(D_k)) \quad (8.14a) \\ C_p(\mathcal{A}) + \max(p_j, \min(P_k)) & \text{if } d_j \leq \delta_p < \max(D_k) \quad (8.14b) \\ C_p(\mathcal{A}) + \max(p_j - \min(P_k), 0) & \text{if } \delta_p \geq \max(D_k) \quad (8.14c) \end{cases}$$

La figure 8.6 illustre le principe de la formule (8.14) : le placement d'une tâche j dans une fournée k implique son transfert de la fournée k depuis un bloc p vers un de ses prédécesseurs ou lui-même, c'est-à-dire un bloc $p' \leq p$ tel que $\delta_{p'} = \min(d_j, \max(D_k))$. Tout d'abord, la séquence de blocs $1, \ldots, p'-1$ n'est pas modifiée par le transfert. Ainsi, les dates d'achèvement et les retards de ces blocs restent identiques. Par conséquent le retard algébrique maximal de cette sous-séquence de blocs est dominé par $L(\mathcal{A})$. En effet, la règle (LF) est appliquée après l'initialisation des blocs, donc avant la règle (AF). Ensuite, les dates d'achèvement des blocs $p', \ldots, p-1$ sont retardées de la durée actualisée $\max(\min(P_k), p_j)$ de la fournée et elles sont donc données par (8.14b). Finalement, l'augmentation de la durée $\max(p_j - \min(P_k), 0)$ de la fournée k est ajoutée aux dates d'achèvement des blocs $p, \ldots, n$ en (8.14c). Si cette augmentation est nulle, alors le retard algébrique maximal des blocs $p, \ldots, n$ est dominé par $L(\mathcal{A})$

L'algorithme réduit le domaine des variables B_j en appliquant la règle (AF). Le filtrage est illustré en figure 8.7 pour le placement de la tâche 4 en partant de l'affectation partielle $\mathcal{A}_2$ (voir figure 8.1 page 81). Après l'initialisation, la boucle L1 parcourt les blocs en ordre inverse pour détecter les placements non réalisables. Le retard algébrique maximal $\Lambda_{\geq p}$ et l'augmentation maximale autorisée de la durée Δ de la sous-séquence de blocs $p, \ldots, n$ sont actualisés grâce à des formules récursives déduites de la section 8.2.1.

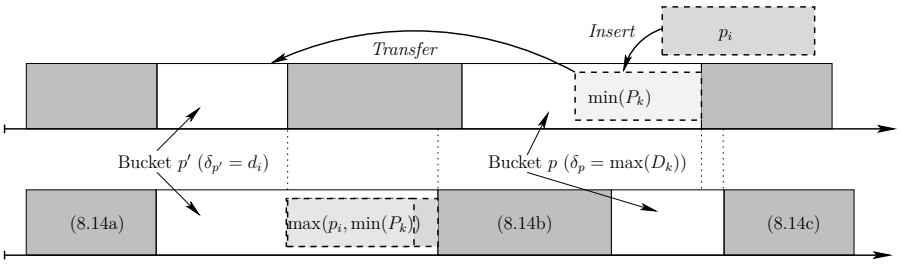
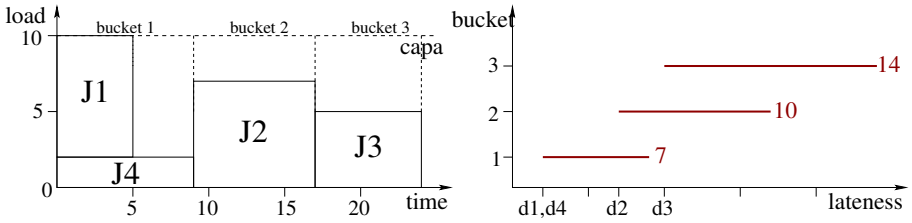


FIGURE 8.6 – Anticipation de l'effet du placement d'une tâche sur la séquence de blocs.

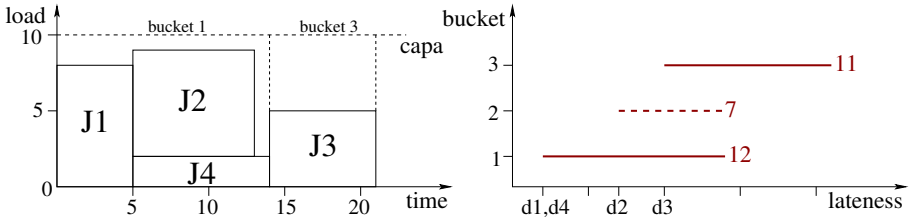
Procédure jobCBF : Filtrage basé sur le coût du placement des tâches en temps quadratique.

$\mathcal{B} = \emptyset$; // Set of batches to prune
$\Lambda_{\geq p} = -\infty$; // Lateness of sub-sequence $p, \ldots, n$
foreach $k \in K$ **do** $\Lambda^k_{\geq p} = -\infty$; // Lateness without batch k of sub-sequence $p, \ldots, n$
/* Try assignments in decreasing order of blocs */
L1 **for** $p \leftarrow n$ **to** 1 **do**
  $\Lambda_{\geq p} = \max(L_p(\mathcal{A}), \Lambda_{\geq p})$; // Maximal lateness of sub-sequence $p, \ldots, n$
  $\Delta = \max(L_{max}) - \Lambda_{\geq p}$; // Maximal duration increase allowed at bloc p
  $\mathcal{B}_{=p} = \{k \in K_{=p} \mid \min(P_k) > 0 \wedge (|\mathcal{D}(P_k)| > 1 \vee |\mathcal{D}(D_k)| > 1)\}$; // New batches to prune
  /* Update durations of new batches at bloc p */
L2 **foreach** $k \in \mathcal{B}_{=p}$ **do** $\max(P_k) \leftarrow \min(P_k) + \Delta$;
  /* assignments of an available job j to other batches */
L3 **foreach** $k \in \mathcal{B}$ **do** $\Lambda^k_{\geq p} = \max(L_p(\mathcal{A}), \Lambda^k_{\geq p})$;
L4 **forall the** $j \in J_{=p}$ **such that** $|\mathcal{D}(B_j)| > 1$ **do**
    /* Assignment to non-empty batches */
L5 **foreach** $k \in \mathcal{B}$ **do**
      if $\delta_p < \min(D_k)$ **then** $\mathcal{B} = \mathcal{B} \backslash \{k\}$;
      else if $\Lambda^k_{\geq p} + \max(\min(P_k), p_j) > \max(L_{max})$ **then** $B_j \not\leftarrow k$;
    /* Assignment to empty batches */
    if $p_j > \Delta$ **then** $\max(B_j) \leftarrow |K^\star|$;
  /* Update data structure */
L6 **foreach** $k \in \mathcal{B}_{=p}$ **do** $\Lambda^k_{\geq p} = \Lambda_{\geq p} - \min(P_k)$;
  $\mathcal{B} = \mathcal{B} \cup \mathcal{B}_{=p}$;

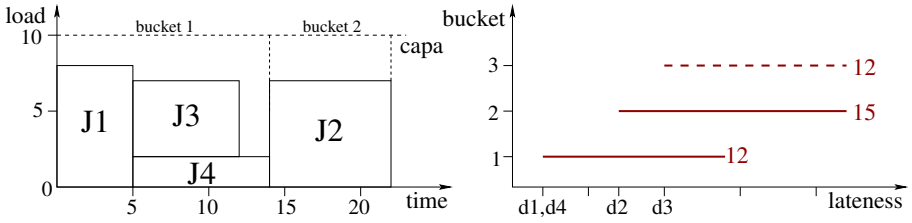
Les fournées de $\mathcal{B}_{=p}$ ordonnancées au plus tard dans le bloc p qui ne sont ni vides ni pleines, sont maintenant examinées par l'algorithme. La boucle L2 actualise la durée maximale autorisée de ces fournées, mais la suppression des placements non réalisables est déléguée aux contraintes (8.1). Cette boucle détecte simultanément tous les placements non réalisables d'une tâche j dans une fournée k tels que $\max(D_k) \leq d_j$. En effet, leurs coûts marginaux ne dépendent que de l'augmentation de la durée de la fournée, car (8.14b) n'est pas défini. Ce cas de figure est illustré en figure 8.7(a).
La boucle L3 actualise incrémentalement le retard algébrique maximal $\Lambda^k_{\geq p}$ de la séquence $p, \ldots, n$ sans la fournée k. Puis, la boucle L4 examine seulement les placements de tâches qui entraînent effectivement le transfert d'une fournée vers le bloc p. La boucle interne L5 examine donc les fournées découvertes dans des blocs précédemment visités par l'algorithme. Si le transfert de la fournée dans le bloc courant n'est pas réalisable à cause des contraintes portant sur les dates échues, alors la fournée est supprimée de l'ensemble $\mathcal{B}$ des fournées à examiner. Dans le cas contraire, le placement de la tâche j dans la fournée k est éliminé lorsque le retard algébrique maximal après le transfert de la fournée dans le bloc p dépasse la meilleure borne supérieure connue. En fait, le calcul incrémental de $\Lambda^k_{\geq p}$ est crucial dans cet algorithme, car il réduit la complexité de la formule (8.14b) d'un temps linéaire à un temps constant. Dans la figure 8.7(b), le transfert de la fournée 2 depuis le bloc 2 vers le bloc 1 ne change pas la séquence de fournées dans la solution de la relaxation, alors que le transfert de la fournée 3 du bloc 3 vers le bloc 1 inverse les positions des fournées 2 et 3 dans la figure 8.7(c).
Ensuite, l'algorithme examine simultanément tous les placements d'une tâche dans une fournée vide pour lesquels les formules (8.14b) and (8.14c) sont égales, car l'augmentation de la durée de la fournée est égale à sa durée ($\min(P_k) = 0$) comme illustré en figure 8.7(d). Dans ce cas, le domaine de la variable B_j est réduit aux valeurs inférieures à l'indice $|K^\star|$ de la dernière fournée non vide.
Finalement, la boucle L6 initialise les retards algébriques maximaux $\Lambda^k_{\geq p}$ des fournées du bloc p en soustrayant leur contribution au retard de la sous-séquence. Ensuite, l'ensemble $\mathcal{B}$ des fournées à examiner est mis à jour.
La complexité de la procédure dépend uniquement de ces boucles puisque toutes les instructions sont exécutées en temps constant. Un calcul simple montre que la complexité ne dépend que des boucles imbriquées L1, L4 et L5. Dans le pire cas, les boucles imbriquées L1 et L4 parcourent une partition de l'ensemble des tâches J. Ce parcours a une complexité $O(n)$. La boucle L5 est réalisée en $O(m)$, car $\mathcal{B} \subseteq K$ est un sous-ensemble de fournées. Par conséquent, la complexité de l'algorithme est en $O(nm)$.



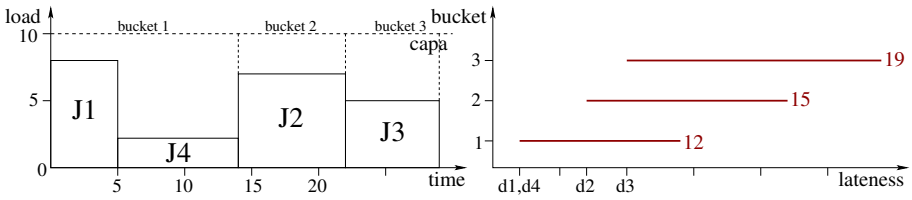
(a) Placement : $B_4 \leftarrow 1$



(b) Placement : $B_4 \leftarrow 2$



(c) Placement : $B_4 \leftarrow 3$



(d) Placement : $B_4 \leftarrow 4$

FIGURE 8.7 – Solution de la relaxation après le placement de la tâche 4 dans une fournée pour l'affectation partielle $\mathcal{A}_2 = \{B_1 \leftarrow 1, B_2 \leftarrow 2, B_3 \leftarrow 3, B_4 \in [1, 4]\}$.