Ordonnancement sur les machines à traitement par batches et contraintes de compatibilité.

Adrien Bellanger

▶ **To cite this version:**

Adrien Bellanger. Ordonnancement sur les machines à traitement par batches et contraintes de compatibilité.. Informatique [cs]. Institut National Polytechnique de Lorraine - INPL, 2009. Français. NNT : . tel-00606383

HAL Id: tel-00606383

https://theses.hal.science/tel-00606383

Submitted on 7 Jul 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Nancy-Université

INPL

AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.
Il est soumis à la propriété intellectuelle de l'auteur au même titre que sa version papier. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.
D'autre part, toute contrefaçon, plagiat, reproduction illicite entraîne une poursuite pénale.

Contact SCD INPL: mailto:scdinpl@inpl-nancy.fr

LIENS

Code de la propriété intellectuelle. Articles L 122.4
Code de la propriété intellectuelle. Articles L 335.2 – L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

Nancy-Université
INPL

Institut National
Polytechnique de Lorraine

Département de formation doctorale en informatique

École doctorale IAEM Lorraine

Ordonnancement sur les machines à traitement par batches et contraintes de compatibilité.

THÈSE

présentée et soutenue publiquement le 23 novembre 2009

pour l'obtention du

Doctorat de l'Institut National Polytechnique de Lorraine

(spécialité informatique)

par

Adrien Bellanger

Composition du jury

Président :	Christian PRINS	*Professeur, Université de Technologique, Troyes*
Rapporteurs :	Pierre LOPEZ	*Chargé de recherche CNRS, Toulouse (HDR)*
	Bernard PENZ	*Professeur, Grenoble INP*
Examinateurs :	Jean-Charles BILLAUT	*Professeur, Polytech'Tours*
	Emmanuel JEANNOT	*Chargé de recherche INRIA, Bordeaux (HDR)*
	Francis SOURD	*SNCF - Direction Innovation & Recherche (HDR)*
	Marie-Claude PORTMANN	*Professeur, École des Mines de Nancy* *Directeur de thèse*
	Ammar OULAMARA	*Maître de conférence, École des Mines de Nancy* *Co-directeur de thèse (HDR)*

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Loria

Mis en page avec la classe thloria.

Remerciements

Ce travail a été financé par une *allocation à la formation recherche (AFR)* du *Fonds national de la recherche (FNR)* du **Luxembourg**. Je tiens donc à remercier le gouvernement du Grand Duché de Luxembourg pour m'avoir permis d'effectuer ces travaux de recherches dans de bonnes conditions.

Par ailleurs, je tiens avant tout à remercier Ammar Oulamara sans qui je n'aurais pas pu mener à bien cette thèse. En effet, non content de m'avoir accordé sa confiance durant ces 3 années, il m'a appris à structurer mes idées, en étant, notamment, à l'écoute de mes pensées les plus confuses. Je tiens également à le remercier pour m'avoir entraîné sur des pistes de travail très intéressantes, et pour s'être fortement investi dans cette thèse. Je souhaite également remercier Marie-Claude Portmann pour les nombreuses pistes d'amélioration qu'elle m'a proposées, pour son regard critique sur le travail effectué et sur les orientations à envisager. Je les remercie également pour leur disponibilité, leurs conseils avisés - scientifiques comme professionnels - et leur bonne humeur.

De plus, je remercie les rapporteurs, Pierre Lopez et Bernard Penz, pour leur lecture attentive de ce manuscrit et pour leurs rapports détaillés et pertinents. Je remercie également les autres membres du jury, Jean-Charles Billaut, Emmanuel Jeannot, Christian Prins et Francis Sourd qui ont accepté d'évaluer mon travail.

Je tiens également à remercier les collègues de l'équipe ORCHIDS pour la bonne ambiance dans l'équipe : merci donc, à Wahiba, Suzana, Khalida, Zerouk, Henri et Ayman sans oublier Françoise Laurent pour son aide précieuse dans les démarches administratives.

Je n'aurais pas pu effectuer cette thèse sans le précieux soutien de ma famille. Merci Claudia. Merci papa et maman, Loulou, Titi et Jojo. Merci aux papys et mamies.

Merci Arnaud et Julie pour votre présence amicale durant ces années de collocation. Je tiens également à remercier les copains handballeurs pour les bons moments passés ensemble. Et surtout un grand merci aux Fraisiens pour leur amitié qui perdure depuis tant d'années.

Table des matières

Introduction générale **1**

Chapitre 1 Le problème et ses applications **3**

1.1 La gestion de production 4

1.2 L'ordonnancement de la production 5

1.2.1 Les problèmes d'ordonnancement 5

1.2.2 Typologie des ordonnancements de la production 5

1.2.3 Représentation des problèmes d'ordonnancement 7

1.3 Problèmes étudiés 8

1.3.1 Flowshop hybride 8

1.3.2 Les machines à traitement par batches 8

1.3.3 Batches et compatibilité 9

1.4 Application industrielle 10

1.4.1 Processus de fabrication d'un pneu 11

1.4.2 Description de l'atelier de production 11

1.4.3 Du problème industriel au problème scientifique 12

Chapitre 2 Modèles et travaux antérieurs **13**

2.1 Classification des problèmes et méthodes de résolution 14

2.1.1 Notations des problèmes d'ordonnancement présentés 14

2.1.2 Classification des problèmes d'ordonnancement 14

2.1.3 Classification de méthodes de résolution 18

2.2 Machines à traitement par batch 22

2.2.1 Problèmes sans contraintes de compatitibilité 23

2.2.2 Problèmes avec contraintes de compatibilité 31

2.3 Flowshop Hybride classique 35

2.4 Flowshop et batches 36

2.5 Conclusion du chapitre 39

Chapitre 3 Méthodes approchées **41**

3.1 Heuristiques . . . 42
3.1.1 Schémas génériques des heuristiques proposées . . . 42
3.1.2 Bornes inférieures . . . 44
3.1.3 Heuristiques pour machines parallèles à traitement par batches . . . 45
3.1.4 Heuristiques pour le flowshop hybride à deux étages . . . 49
3.1.5 Résultats expérimentaux . . . 62
3.2 Durées d'exécution identiques au premier étage . . . 68
3.2.1 Principe . . . 68
3.2.2 Transformation des données . . . 69
3.2.3 Ordonnancement des tâches courtes . . . 72
3.2.4 Ordonnancement des tâches longues . . . 73
3.2.5 Schéma d'approximation polynomial (PTAS) . . . 74
3.3 Conclusion du chapitre . . . 76

Chapitre 4 Méthodes Exactes **77**

4.1 Méthodes par séparation évaluation pour le flowshop hybride . . . 78
4.1.1 Méthode de Carlier et Néron . . . 78
4.1.2 Adaptation au problème de flowshop hybride avec machines à traitement par batches au second étage . . . 79
4.2 Méthode directe . . . 80
4.2.1 Première étape . . . 80
4.2.2 Seconde étape. . . . 83
4.3 Méthode inverse . . . 85
4.3.1 Première étape . . . 85
4.3.2 Seconde étape. . . . 88
4.4 Résultats Expérimentaux . . . 90
4.4.1 Performances des heuristiques . . . 90
4.4.2 Comparaison des bornes inférieures . . . 91
4.4.3 Comparaison des deux méthodes exactes . . . 93
4.5 Conclusion du chapitre . . . 103

Chapitre 5 Autres critères réguliers **105**

5.1 Propriétés des ordonnancements optimaux . . . 106
5.2 Minimisation du Makespan . . . 107
5.3 Minimisation de la somme des dates de fin d'exécution . . . 107
5.3.1 Nombre de batches reportés . . . 108

5.3.2 Algorithme de programmation dynamique 110
5.4 Minimiser des fonctions objectif avec dates de fin souhaitées 112
5.4.1 Algorithme de programmation dynamique 115
5.5 Conclusion du chapitre 117

Chapitre 6 Conclusion et perspectives **119**
6.1 Conclusion 120
6.2 Perspectives 121
6.2.1 Intensification des expériences 121
6.2.2 Extension des modèles étudiés 121
6.2.3 Autres perspectives 122

Annexes **125**

Annexe A Résultats expérimentaux détaillés des heuristiques. **125**

Annexe B Résultats expérimentaux des méthodes exactes pour des instances de 15 tâches. **131**

Bibliographie **135**

Glossaire des notations

Notations des problèmes

n : le nombre de tâches.
J : l'ensemble des tâches du problème.
$\overline{J}$: l'ensemble de tâches non ordonnancées à l'itération courante d'une méthode de résolution.
$|\overline{J}|$: le nombre de tâches de l'ensemble $\overline{J}$.

m_s : le nombre de machines de l'étage s.
m : le nombre maximum de machines sur un étage ($m = \max_s m_s$), si seulement un étage possède plusieurs machines, m désigne le nombre de machines de cet étage.

p_{ij} : la durée d'exécution de la tâche d'indice j à l'étage i, il est à noter qu'en absence d'ambiguïté sur l'étage désigné (un seul étage ou durées identiques sur chaque étage) la valeur de i n'est pas indiquée.
$[a_{ij},\ b_{ij}]$: l'intervalle de durée d'exécution de la tâche d'indice j à l'étage i, en l'absence d'ambiguïté l'intervalle est noté $[a_j, b_j]$. Cet intervalle est dit *proportionnel*, si l'ordonnée du point final de l'intervalle dépend de celle du point initial alors l'intervalle est noté $[a_j, (1+\alpha)a_j]$.
a_{max} : la plus grande des ordonnées des points initiaux d'intervalles de durées d'exécution $[a_j, b_j]$ ou $[a_j, (1+\alpha)a_j]$ ($a_{max} = \max_{j\in J}\{a_j\}$).
s_{ikj} : le temps de réglage de la tâche d'indice j sur l'étage i si elle succède à la tâche d'indice k. Si ce temps de réglage ne dépend que de la tâche d'indice j, il est noté s_{ij}. En l'absence d'ambiguïté nous utilisons s_{kj} et s_j.
q_{ij} : le temps de finition de la tâche d'indice j sur l'étage i. En l'absence d'ambiguïté nous utilisons q_j.
r_j : la date de disponibilité de la tâche d'indice j.
$\tilde{r}_i$: la $i^{\text{ème}}$ date de disponibilité généralisée, et $nb_{\tilde{r}_i}$ le nombre de tâches disponible à cette date.
$\tilde{r}_{max}$: désigne la plus grande date de disponibilité généralisée ($\tilde{r}_{max} = \max\{\tilde{r}_i\}$).
d_j : la date de fin d'exécution souhaitée de la tâche d'indice j.
$\tilde{d}_j$: la date de fin d'exécution impérative de la tâche d'indice j.
v_{ij} : le volume (ou taille) de la tâche d'indice j à l'étage i, en l'absence d'ambiguïté elle est noté v_j.

B_i : le batch d'indice i.
$P(B_i)$: la durée d'exécution du batch B_i.
$|B_i|$: le nombre de tâches du batch B_i.
k_i : la capacité des machines à traitement par batches de l'étage i. À l'instar de la notation

des durées d'exécution en l'absence d'ambiguïté la valeur de i n'est pas indiquée.

b^* : le nombre maximum de tâches compatibles avec une même durée d'exécution, c'est-à-dire la taille maximale d'un batch lorsque la capacité est infinie.

t_m^i : la date de disponibilité de la machine m sur l'étage i.

w_j : le poids associé à la tâche d'indice j dans la fonction objectif.
C_j : la date de fin d'exécution de la tâche d'indice j.
$L_j = C_j - d_j$: le retard algébrique de la tâche d'indice j.
$T_j = \max\{C_j - d_j; 0\}$: le retard de la tâche d'indice j.
$U_j = \{ {}^{1 \ si \ C_j > d_j}_{0 \ sinon}$: indicateur de retard de la tâche d'indice j.

Table des figures

1.1 Ordonnancement d'un flowshop hybride 7
1.2 Étapes étudiées du système de production d'un pneu 12

3.1 Ordonnancement du problème initial (inversion des étages). 43
3.2 Ordonnancement du problème inverse (inversion des étages). 43
3.3 Graphe de compatibilité (FCBLPT) 45
3.4 Ordonnancement optimal (FCBLPT). 45
3.5 Ordonnancement $S_{H_{List}}$ (pour Bm_2). 47
3.6 Ordonnancement optimal S^* (H_{List} pour Bm_2). 47
3.7 Ordonnancement $S_{H_{LPT}}$ (pour Bm_2). 48
3.8 Ordonnancement optimal S^* (H_{LPT} pour Bm_2). 49
3.9 Ordonnancement $S_{H_{LPT}}$ (pour $HF2(m_1, Bm_2)$). 51
3.10 Ordonnancement optimal S^* (H_{LPT} pour $HF2(m_1, Bm_2)$). 52
3.11 Ordonnancement $S_{H_{LBPT}}$ (pour $HF2(1, Bm)$). 57
3.12 Ordonnancement optimal S^* (H_{LBPT} pour $HF2(1, Bm)$). 57
3.13 Ordonnancement $S_{H_{LPT}}$ (pour $HF2(m, B)$). 60
3.14 Ordonnancement optimal S^* (H_{LPT} pour $HF2(m, B)$). 60
3.15 Ordonnancement S_{H_J} (pour $HF2(m, B)$). 62
3.16 Ordonnancement optimal S^* (H_J pour $HF2(m, B)$). 62

4.1 Exemple d'arbre de recherche pour la méthode directe (PSE). 80

5.1 Structure d'un ordonnancement réalisable 113

Liste des tableaux

2.1 Résumé des notations des méthodes utilisées dans l'état-de-l'art. 23
2.2 Complexité des problèmes à une machine à traitement par batches, avec capacité infinie, sans date de disponibilité ($B|b \geqslant n|f$). 24
2.3 Complexité des problèmes à une machine à traitement par batches, avec capacité finie, sans date de disponibilité ($B|b = k|f$). 24
2.4 Complexité des problèmes à une machine à traitement par batches avec dates de disponibilité et capacité infinie ($B|b \geqslant n, r_j|f$). 25
2.5 Complexité des problèmes à une machine à traitement par batches avec dates de disponibilité et capacité finie ($B|b = k, r_j|f$). 25
2.6 Complexité des problèmes à une machine à traitement par batches avec restriction sur les durées d'exécution et dates dues. 27
2.7 Complexité des problèmes à une machine à traitement par batches (cas particuliers) 28
2.8 Complexité des problèmes à une machine à traitement par batches avec volumes des tâches variables (v_j) 29
2.9 Complexité des problèmes à plusieurs machines parallèles à traitement par batches sans compatibilité 30
2.10 Complexité des problèmes à une machine à traitement par batches avec familles incompatibles entre elles. 32
2.11 Complexité des problèmes à une machine à traitement par batches avec graphe de compatibilité 33
2.12 Complexité des problèmes à plusieurs machines parallèles à traitement par batches avec compatibilité 34
2.13 Complexité des problèmes de flowshops hybrides avec machines discrètes 35
2.14 Complexité des problèmes combinant flowshop et machines à traitement par batch 37
2.15 Complexité des problèmes combinant flowshop hybride et machines à traitement par batch 38

3.1 Durées d'exécution des tâches (inversion des étages). 43
3.2 Durées d'exécution des tâches (FCBLPT). 45
3.3 Durées d'exécution des tâches de l'instance I (H_{List} pour Bm_2). 47
3.4 Durées d'exécution des tâches de l'instance I (H_{LPT} pour Bm_2). 48
3.5 Durées d'exécution des tâches de l'instance I (H_{LPT} pour $HF2(m_1, Bm_2)$). . . . 51
3.6 Durées d'exécution des tâches de l'instance I (H_{LPBT} pour $HF2(1, Bm)$). 56
3.7 Durées d'exécution des tâches de l'instance I (H_{LPT} pour $HF2(m, B)$). 60
3.8 Durées d'exécution des tâches de l'instance I (H_J pour $HF2(m, B)$). 61
3.9 Comparaison des heuristiques pour $HF2(m_1, Bm_2)$. 66
3.10 Comparaison des heuristiques pour $HF2(m_1, Bm_2)$ (moyenne globale). 67

4.1 Comparaison des heuristiques par rapport à l'optimum (PSE). 91
4.2 Comparaison des durées d'exécution pour chaque borne de la première étape de la méthode directe (PSE). 92
4.3 Comparaison des durées d'exécution de la borne LBD_1^p pour 15 tâches (PSE). . . 92
4.4 Comparaison des durées d'exécution de la borne LBD_1^p pour 20 tâches (PSE). . . 93
4.5 Comparaison des durées d'exécution pour chaque borne de la seconde étape de la méthode directe (PSE). 93
4.6 Comparaison des durées d'exécution pour chaque borne de la méthode inverse (PSE). 94
4.7 Comparaison des PSE lorsqu'il y a 2 machines sur chaque étage. 98
4.8 Comparaison des PSE lorsqu'il y a 2 machines sur le premier étage et 5 sur le second. 99
4.9 Comparaison des PSE lorsqu'il y a 5 machines sur le premier étage et 2 sur le second. 100
4.10 Comparaison des PSE lorsqu'il y a 5 machines sur le premier étage et 5 sur le second. 101
4.11 Comparaison des PSE en fonction de n. 102

A.1 Comparaison des heuristiques pour $HF2(m_1, Bm_2)$ lorsque $c_2 = 1$. 127
A.2 Comparaison des heuristiques pour $HF2(m_1, Bm_2)$ lorsque $c_2 = 2$. 128
A.3 Comparaison des heuristiques pour $HF2(m_1, Bm_2)$ lorsque $c_2 = 3$. 129
A.4 Comparaison des heuristiques pour $HF2(m_1, Bm_2)$ lorsque $c_2 = 4$. 130

B.1 Comparaison des PSE pour m_1-m_2 =2-2 et 5-5 lorsqu'il y a 15 tâches. 132
B.2 Comparaison des PSE pour m_1-m_2 =2-5 et 5-2 lorsqu'il y a 15 tâches. 133

Introduction générale

Cette thèse, financée par le «Fonds National de la Recherche» (Luxembourg), a été effectuée au sein de l'équipe ORCHIDS (Operations Research for Complex HybrId Decision Systems) du LORIA (Laboratoire lOrrain de Recherche en Informatique et ses Applications) sous la direction de Marie-Claude Portmann et Ammar Oulamara. Les recherches effectuées au sein de l'équipe ORCHIDS concernent la recherche opérationnelle, principalement des problèmes d'ordonnancements et des problèmes liés aux chaînes logistiques.

Originellement, en théorie de l'ordonnancement d'atelier les problèmes étudiés sont des problèmes avec machines disjonctives, c'est-à-dire que les machines traitent une seule tâche à la fois. Cependant de nombreuses machines rencontrées dans l'industrie permettent de traiter plusieurs tâches simultanément. Depuis une vingtaine d'années, la communauté de recherche en ordonnancement de la production s'intéresse également à ce type de machines appelées *machines à traitement par batches* (de type *p-batch*). Ces machines permettent d'exécuter plusieurs tâches simultanément au sein d'un *batch* (lot). La plupart des travaux de recherche sur machines à traitement par batches sont inspirés de problèmes réels rencontrés dans l'industrie du semi-conducteur et de la métallurgie, mais ce type de machines est également rencontré dans d'autres secteurs d'activités tels l'industrie de pneumatiques, l'industrie pharmaceutique ou encore la cristallerie. En pratique, des incompatibilités entre les tâches ne permettent pas certaines combinaisons de tâches dans un même batch. Ces incompatibilités peuvent être liées, par exemple, aux composants des différents produits, à leurs couleurs, ou à leurs températures de cuisson. Dans le cas qui nous intéresse dans cette thèse, les tâches ont des durées d'exécution minimales et maximales, c'est pourquoi les tâches d'un même batch doivent partager un même ensemble de durées d'exécution possibles.

La majeure partie de cette thèse est consacrée aux problèmes de minimisation de la date de fin d'exécution d'un flowshop hybride à deux étages avec des machines à traitement par batches et compatibilité entre les tâches sur le second étage. Un flowshop hybride à 2 étages est un atelier de production dans lequel les tâches sont exécutées sur l'une des machines du premier étage, puis lorsque leur exécution est complétée sur le premier étage elles sont exécutées sur le second étage.

Le traitement de ces problèmes nous a été inspiré par une entreprise de pneumatiques. La construction d'un pneumatique est principalement articulée autour de deux phases : la phase d'assemblage qui est composée de machines qui assemblent les pneus les uns après les autres, et la phase de cuisson, où les pneus assemblés sont cuits dans des machines à presse qui peuvent contenir plusieurs pneus. Ainsi, les presses sont des machines à traitement par batches avec des contraintes supplémentaires de compatibilité entre les tâches.

Le premier chapitre de cette thèse expose les généralités des problématiques d'ordonnancement de la production. Ainsi, la typologie présentée dans ce préambule nous permet de définir les problèmes abordés dans cette thèse, et de présenter l'application industrielle qui nous a inspiré l'étude de ces problèmes.

Dans le second chapitre, la présentation des classifications des problèmes et méthodes simplifie les descriptions des travaux présents dans la littérature concernant à la fois les machines à traitement par batches et les flowshops. À la lecture de cet état-de-l'art nous remarquons que les problèmes étudiés dans cette thèse ont été très peu abordés jusqu'à présent.

Suite à la présentation des problèmes et des travaux existant dans la littérature, nous développons, dans le troisième chapitre, des méthodes approchées afin de résoudre deux problèmes de minimisation de la date de fin de l'ordonnancement pour un flowshop avec machines à traitement par batches et compatibilité entre les tâches. Une première série de six méthodes permet de résoudre approximativement le problème général et les cas particuliers où l'un des deux étages contient seulement une machine. Ces méthodes constructives, que nous appelons *heuristiques*, sont à performance garantie, c'est-à-dire que nous avons une borne supérieure de l'écart entre la solution optimale et la solution obtenue par l'heuristique avant même l'exécution de cette heuristique. La dernière méthode approchée que nous avons développée permet de résoudre le problème de flowshop hybride lorsque les durées d'exécution du premier étage sont identiques. Cette méthode est de type schéma d'approximation polynomial (PTAS). Elle fournie une solution approchée avec une garantie de performance, et la durée d'exécution de la méthode dépend de la garantie de performance souhaitée.

Dans le quatrième chapitre, nous développons deux méthodes exactes arborescentes pour résoudre le problème de minimisation de la date de fin d'ordonnancement pour un flowshop hybride avec machines à traitement par batches et compatibilité entre les tâches. Alors que la première méthode traite le problème avec machines disjonctives sur le premier étage, la seconde méthode traite le problème inverse, c'est-à-dire que les machines disjonctives sont sur le second étage, et les machines à traitement par batches sur le premier.

Finalement le cinquième chapitre, s'intéresse à la minimisation de critères réguliers sur une machine à traitement par batches de capacité infinie et compatibilité entre les tâches. Nous présentons la complexité de ces différents problèmes et quelques méthodes permettant de les résoudre.

Un chapitre de conclusion, présentant quelques perspectives intéressantes de développement des travaux présentés ici, achève cette thèse.

Chapitre 1

Le problème et ses applications

Ce chapitre est consacré à la présentation des problèmes traités dans cette thèse. La première section définit, de manière concise, la gestion de production. Ceci permet de situer les problèmes d'ordonnancement de la production, présentés dans la seconde section, au sein de la gestion d'une entreprise. Ensuite, nous présentons les problèmes d'ordonnancement de la production traités dans ce manuscrit. La dernière section présente l'application industrielle qui nous a incitée à étudier ces problèmes.

Sommaire

1.1 La gestion de production 4
1.2 L'ordonnancement de la production 5
1.2.1 Les problèmes d'ordonnancement 5
1.2.2 Typologie des ordonnancements de la production 5
1.2.3 Représentation des problèmes d'ordonnancement 7
1.3 Problèmes étudiés 8
1.3.1 Flowshop hybride 8
1.3.2 Les machines à traitement par batches 8
1.3.3 Batches et compatibilité 9
1.4 Application industrielle 10
1.4.1 Processus de fabrication d'un pneu 11
1.4.2 Description de l'atelier de production 11
1.4.3 Du problème industriel au problème scientifique 12

1.1 La gestion de production

La *production* désigne l'utilisation de ressources, en vue de créer des biens ou des services.

Lors de cette création, des matières premières (et/ou des produits semi-finis) sont transformés en produits finis. L'objectif de toute production est de satisfaire la demande des clients avec des produits de qualité garantie fabriqués dans les délais impartis, et ceci à moindre coût. Cette fabrication se fait à l'aide d'un ensemble de moyens appelé *système de production*. Le système de production regroupe un ensemble de ressources telles que les hommes et les machines, mais aussi les capacités de stockage, les informations techniques. La *gestion de production* désigne l'activité de gestion de l'ensemble du système de production.

L'objectif de la *gestion de production* est de gérer ce système de production au mieux. Les décisions liées à cette problématique sont habituellement classées suivant 3 catégories introduites par Anthony [13] : les décisions stratégiques, tactiques et opérationnelles. Ces décisions sont prises à des niveaux hiérarchiques différents, avec des granularités et des horizons temporels différents.

- *Les décisions stratégiques* définissent la politique à long terme de l'entreprise (souvent à plus de deux ans), ces décisions portent principalement sur le portefeuille d'activités que l'entreprise souhaite posséder à long terme, ainsi que sur les ressources stables nécessaires à la concrétisation de ses objectifs. Les ressources stables concernées par ces décisions sont les hommes (compétences à posséder, hors intérim) et les machines, mais également les informations de production contenues dans le système d'information. Ces décisions portant sur la stratégie de l'entreprise sont prises par les dirigeants de l'entreprise (généralement avec l'aide de quelques cadres compétents) et portent sur des centaines ou milliers d'heures de travail par famille de produits.
- *Les décisions tactiques* définissent la politique à moyen terme de l'entreprise, elles portent sur la planification de la production, c'est-à-dire la programmation prévisionnelle de la production mensuelle (ou hebdomadaire selon l'entreprise et les types de produits) de chaque famille de produits. Cette planification est généralement établie pour une période allant de 6 à 18 mois. Notons que si les tournées d'approvisionnement et de distribution sont relativement stables, alors des tournées types sont également utilisées. L'horizon de temps étant trop court pour pouvoir effectuer des modifications conséquentes des capacités de productions, ces décisions sont contraintes par les décisions stratégiques. Les décisions tactiques sont du ressort des cadres.
- *Les décisions opérationnelles* permettent d'assurer la production au quotidien, afin de réagir aux fluctuations de la demande et de capacité des ressources prévues ainsi qu'aux aléas. Les décisions opérationnelles concernant la gestion de la production comprennent la gestion des stocks et l'ordonnancement de la production, ce dernier consiste en une affectation prévisionnelle détaillée des ressources à des tâches afin de fabriquer les produits définis par la planification. Généralement ces décisions se prennent sur un horizon de 24, 48 voire 72 heures. En cas de systèmes productifs pilotés, il peut être nécessaire d'employer une granularité temporelle plus fine. Même si la préparation de ces décisions est généralement effectuée par des cadres, les décisions finales sont généralement prises par des agents de maîtrise ou des agents d'exécution.

Dans la suite de ce travail nous nous intéressons uniquement à l'ordonnancement de la production, c'est-à-dire au niveau des décisions opérationnelles.

1.2 L'ordonnancement de la production

1.2.1 Les problèmes d'ordonnancement

Ordonnancer c'est allouer les ressources à des tâches dans le but d'atteindre et/ou d'optimiser un ou plusieurs objectifs.

Aux vues de cette définition très générale, nous remarquons que les problèmes d'ordonnancement sont présents non seulement dans les décisions opérationnelles de gestion de la production mais également dans de nombreux autres domaines. Les problématiques (et méthodes de résolution) de ces problèmes étant diverses et variées, nous séparons les problèmes d'ordonnancement en quatre familles principales.

- *Les ordonnancements temps réels* sont à la base de tout système temps réel, c'est-à-dire tout système dont le comportement s'adapte aux modifications de situations. Un problème d'ordonnancement en temps réel est caractérisé par la connaissance des tâches uniquement lorsqu'elles sont disponibles. Ils comportent en général deux grandes familles de tâches, les tâches récurrentes auxquelles sont associées des fréquences et des fenêtres temporelles à planifier, et des tâches accidentelles prioritaires. Ces problèmes sont présents dans les systèmes embarqués, les systèmes de surveillance, l'allocation de ressources par le système d'exploitation d'un ordinateur...
- *Les ordonnancements de grille* permettent d'ordonnancer les tâches dans les grilles de calculs, ou systèmes distribués, c'est-à-dire tout système de calcul sur plusieurs machines avec des fluctuations du nombre et de la qualité des machines. Un problème d'ordonnancement de grille est donc caractérisé par des incertitudes concernant l'évolution des ressources.
- *Les ordonnancements de projets* permettent de planifier la réalisation de projets. Ils sont caractérisés par un nombre relativement réduit de tâches, compensés par un grand nombre de ressources et de contraintes. Contrairement aux deux familles de problèmes précédentes toutes les données sont supposées connues a priori dans la phase prévisionnelle. L'ordonnancement de projet est utilisé lors de la planification de la construction d'ouvrages d'art tel le viaduc de Millau, ou le tunnel sous la Manche, mais également lors du développement de nouveaux produits (véhicules, logiciels...).
- *Les ordonnancements de production,* que nous considérons dans cette thèse, représentent une partie des décisions opérationnelles de la gestion de la production. Pareillement aux problèmes d'ordonnancement de projet, toutes les données du problème sont supposées connues a priori, en revanche le nombre de ressources est limité et il y a beaucoup de tâches.

Dans ces définitions relativement littéraires des problèmes d'ordonnancement, les termes ressources, tâches et objectifs ne sont pas définis avec précision. Nous reviendrons sur ces définitions dans la section suivante, en les présentant pour les problèmes d'ordonnancement de la production.

1.2.2 Typologie des ordonnancements de la production

A. Les tâches

Une *tâche* est une suite ordonnée ou non de traitements à effectuer sur une pièce ou un lot de pièces insécables (toujours traitées ensemble). En termes plus génériques, une *tâche* est une entité élémentaire de travail qui doit être exécutée sur des ressources du système de production, elle est constituée d'un ensemble d'opérations qui consomment chacune des ressources qui peuvent être différentes d'une opération à l'autre. Chaque opération est localisée dans le temps par des

dates de début et de fin d'exécution et par une durée d'exécution, elle est également localisée dans l'"espace" par les ressources qu'elle consomme et les quantités consommées. Généralement, l'exécution d'une opération est divisée en 4 phases, la préparation (préparation ou remise en état de certaines des ressources, par exemple le réglage des machines, le nettoyage de matières premières, la formation des opérateurs...), la phase principale, la finition (les retouches à effectuer sur la pièce fabriquée, le refroidissement...), et le transport (transport vers le client, vers la prochaine machine...).

Chaque tâche est caractérisée par un ensemble d'opérations (avec la possibilité d'imposer un ordre entre ces opérations), une date de début au plus tôt, et une date de fin souhaitée ou une date de fin impérative. L'exécution de ces tâches peut également être contrainte, par exemple par des contraintes de précédence, c'est-à-dire que l'exécution d'une tâche implique la fin de l'exécution d'une ou plusieurs tâches.

B. Les Ressources

Une *ressource* est un moyen technique ou humain utilisé lors de la réalisation d'une opération. La *capacité* d'une ressource détermine la quantité disponible de cette ressource, nous supposons ici cette capacité constante. Une ressource est dite *renouvelable* (les hommes, les machines, les capacités de stockage...) si elle est de nouveau utilisable avec la même capacité après avoir été utilisée. Au contraire la quantité cumulée d'une ressource consommable (matières premières, produits semi-finis, énergie, argent...) est limitée. Nous effectuons une distinction parmi les ressources renouvelables entre les ressources *disjonctives* qui ne peuvent pas exécuter plusieurs opérations simultanément, et les ressources *cumulatives* qui le peuvent.

C. Les familles d'ateliers (systèmes de production)

La suite de traitements à effectuer pour une tâche, peut être découpée, éventuellement de différentes manières, en une suite ordonnée ou non d'opérations associées chacune à une famille de machines (éventuellement réduite à un seul élément), cette suite est appelée *gamme de fabrication*.

Nous définissons alors différents types d'ateliers dépendant de la manière dont l'ensemble des machines peut-être structuré ou non par les contraintes relatives aux gammes de fabrication des différents produits (ou tâches).

- *Atelier à étage unique* : Les tâches sont composées d'un seul traitement assuré par une seule famille de machines (éventuellement réduite à un seul élément).
- *Atelier de type "openshop"* : Les tâches sont composées d'une suite de traitements assurés chacun par une seule famille de machine (éventuellement réduite à un seul élément), mais cette suite n'est pas ordonnée a priori.
- *Atelier de type "jobshop'* : Les tâches sont composées d'une suite de traitements ordonnés sur une série de famille de machines (éventuellement réduite à un seul élément), la suite dépend du type de pièce.
- *Atelier de type "flowshop"* : Les tâches sont composées d'une suite de traitements assurés chacun par une seule famille de machine (éventuellement réduite à un seul élément), la suite ordonnée ne dépend pas de la pièce, en outre on suppose que les familles de machines sont disjointes. On parle alors d'*étages*.

D. Les critères d'évaluation

Pour compléter la définition d'un problème d'ordonnancement, l'objectif peut-être d'optimiser un critère (minimisation du coût, des pénalités d'avance-retard...) ou bien d'atteindre certains objectifs (toutes les tâches sont ordonnancées dans le mois, aucune tâche n'est en retard...) dans ce dernier cas on dit que l'on cherche un ordonnancement *admissible*. Les critères d'optimisation consistent à *minimiser* ou *maximiser* une *fonction objectif*. Cette fonction objectif peut dépendre de coûts, de durées (plus grande date de fin d'exécution...), du nombre de tâches en retard, du nombre d'interruptions... Dans certains cas, il est nécessaire d'optimiser plusieurs critères, deux possibilités s'offrent alors : pondérer les critères dans une seule fonction objectif à optimiser, ou alors optimiser plusieurs critères simultanément à l'aide de techniques spécifiques à l'ordonnancement multi-critères (front de Pareto...).

Un critère est dit *régulier* si l'avancement d'une tâche de l'ordonnancement ne peut dégrader la valeur de ce critère d'évaluation. Dans cette thèse nous nous intéressons exclusivement aux problèmes d'optimisation de critères réguliers simples (sans combinaison de critères).

E. Représentation graphique d'un ordonnancement

Traditionnellement les ordonnancements sont représentés à l'aide de diagrammes de Gantt, ceux-ci développés en 1910 par H.L. Gantt [80] (qui a notamment travaillé avec F.W. Taylor) à destination de l'ordonnancement de projet ont été adaptés à l'ordonnancement de la production. Ce sont des diagrammes à barres où les opérations sont représentées par des segments (ou rectangles) de longueur proportionnelle à leur durée. En ordonnancement de projet chaque activité est représentée sur une ligne qui lui est propre, alors qu'en ordonnancement de production une ligne correspond à une ressource. La figure 1.1 présente un diagramme de Gantt pour un problème d'ordonnancement sur un flowshop à deux étages, chaque étage possédant 2 machines.

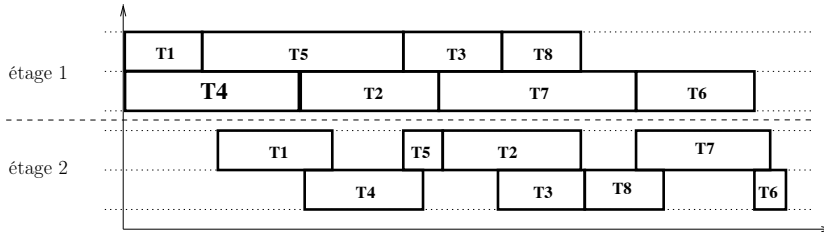


FIGURE 1.1 – Ordonnancement d'un flowshop hybride

1.2.3 Représentation des problèmes d'ordonnancement

La typologie étant introduite, nous remarquons qu'elle permet de définir les problèmes, cependant ces définitions littéraires sont longues et complexes. C'est pourquoi les membres de la communauté de recherche en ordonnancement ont exprimé le besoin, dans les années 70, de définir une notation pratique (simple, logique et concise). Cette notation initiée par Conway et al. [50], puis améliorée par Rinnoy Kan [191], Graham et al. [86] et Blazewicz et al. [21] respectivement en 1967, 1976, 1979 et 1983, a été enrichie par la suite. Cette notation s'articule autour de 3 champs $\alpha|\beta|\gamma$, où α décrit le type d'atelier et les types et nombres de machines, β décrit les caractéristiques des tâches et les contraintes qui leur sont imposées, enfin γ indique le critère d'évaluation de l'ordonnancement.

Ce formalisme, que nous détaillons dans la section 2.1.2, permet ainsi d'utiliser la notation $F2|r_j, p_{2j} = p|C_{max}$ à la place d'une description littéraire de ce problème simple. Cette notation signifie : un problème de minimisation de la plus grande date de fin d'un flowshop à deux étages composé de machines disjonctives avec durées d'exécution identiques au second étage et existence de dates de disponibilité différentes.

1.3 Problèmes étudiés

Les précédentes sections nous ayant permis de mieux appréhender les problèmes d'ordonnancement de la production, nous définissons le problème traité dans cette thèse.

1.3.1 Flowshop hybride

Pour rappel (voir 1.2.2), dans un *flowshop à deux étages* chaque tâche est exécutée au premier étage, puis lorsque son exécution est terminée sur le premier étage elle est exécutée au second étage. On parle de *flowshop hybride à deux étages* lorsqu'il y a plusieurs machines sur au moins l'un des deux étages.

Dans les chapitres 3 et 4 nous considérons un atelier de type flowshop hybride à deux étages avec des machines disjonctives sur le premier étage et des machines cumulatives sur le second étage. Alors que dans le chapitre 5 nous considérons un atelier avec seulement un étage composé d'une seule machine cumulative.

1.3.2 Les machines à traitement par batches

Dans la sous-section précédente nous avons mentionné la présence de machines cumulatives dans tous les problèmes que nous considérons, ces machines sont appelées en ordonnancement de la production *machines à traitement par batches* (ou *lot*). Notons que le terme traitement par batches désigne deux types de traitements différents, le second type de traitement par batches s'effectuant sur des ressources disjonctives. L'intérêt de la communauté scientifique pour ce type de problèmes depuis une quinzaine d'années, s'explique par l'utilisation importante de traitement par batches dans l'industrie et notamment dans les industries chimique, pharmaceutique et des semi-conducteurs.

Après une brève introduction des problèmes de machines à traitement par batches de type s-batch, pour lesquels les ressources sont disjonctives, nous présentons les problèmes de machines à traitement par batches de type p-batch, avec machines cumulatives.

Machines de type s-batch

Une machine de type *sum-batch* ou *serial-batch*, notée aussi *s-batch*, exécute séquentiellement une série de tâches réunies au sein d'un batch (lot). Les tâches d'un tel batch sont exécutées l'une après l'autre sans interruption (réglages...), le réglage qui est commun à toutes ces tâches, est réalisé avant la première tâche du batch. Généralement les tâches utilisant les même réglages sont regroupées en famille et un batch ne peut être composé de tâches de familles différentes. Ainsi entre des tâches d'une même famille il n'y a pas de réglage de la machine mais la machine est réglée entre deux tâches de familles différentes.

Ce type de machines est notamment utilisé dans les processus de fabrication chimique et pharmaceutique, ou encore les ateliers de peinture. En effet dans ces industries, tout changement

de produit sur une chaîne de production implique un nettoyage consciencieux des cuves, des réacteurs... Les articles de Potts et van Wassenhove [188], Neumann et al. [168], Castro et al. [32] fournissent des exemples d'utilisations industrielles de ce type de machines.

Machines de type p-batch

Une machine de type *max-batch* ou *parallel-batch*, notée aussi *p-batch*, exécute plusieurs tâches parallèlement au sein d'un batch. Toutes les tâches d'un tel batch sont exécutées simultanément. C'est-à-dire que la durée d'exécution du batch correspond à celle de la plus grande tâche du batch, toutes les tâches doivent être présentes dès le début du batch et ne sont pas achevées avant la fin du batch. Généralement les machines à traitement par batches (de type p-batch) ont une *capacité* finie, c'est-à-dire qu'elles ne peuvent pas traiter un nombre infini de tâches. Cette capacité peut restreindre le nombre de tâches exécutées dans un batch, mais également leur poids total, leur volume total...

Ce type de machines est grandement utilisé dans l'industrie des semi-conducteurs. En effet, la fabrication de semi-conducteurs est généralement divisée en 4 phases, tout d'abord la fabrication du circuit puis son sondage. Ensuite les circuits sont assemblés sur les cartes électroniques, finalement le produit fini est testé. Ce test s'effectue en soumettant les cartes à différentes températures et voltages dans un four. Lors de cette phase de test plusieurs cartes sont enfournées simultanément, ainsi chaque four est une machine à traitement par batches (p-batch). Mathirajan et Sivakumar [157] fournissent un état-de-l'art de l'utilisation de machines à traitement par batches dans l'industrie de semi-conducteurs. Le problème industriel, qui nous a inspiré (section 1.4), issu de l'industrie du pneumatique, utilise également ce type de machines.

Par la suite nous abordons uniquement les machines de type p-batch, de plus afin de faciliter la lecture nous utilisons le terme de *machine à traitement par batches* à la place de *machine à traitement par batches de type p-batch*. Notons que l'état-de-l'art de Potts et Kovalyov [185] traite les deux types de machines, il permet ainsi d'approfondir cette introduction à l'ordonnancement de machines à traitement par batches de type s-batch.

1.3.3 Batches et compatibilité

La définition de machines à traitement par batches (p-batch) donnée précédemment ne permet pas d'interdire certaines combinaisons de tâches, or en pratique il est, par exemple, impossible de cuire deux objets à des températures différentes dans un même four. C'est pourquoi certaines combinaisons de tâches doivent être interdites, d'où l'intérêt d'ajouter des contraintes de compatibilité entre les tâches. Étant donné que les caractéristiques (durée d'exécution, température de cuisson, matériaux utilisés...) des tâches peuvent être extrêmement différentes, plusieurs types de compatibilités sont utilisés. Ainsi deux tâches sont dites *compatibles* si elles peuvent appartenir à un même batch, c'est-à-dire qu'elles partagent les mêmes caractéristiques. Ainsi toutes les tâches d'un batch doivent être deux à deux compatibles. Une tâche est dite *compatible* avec un batch si elle est compatible avec toutes les tâches du batch.

Nous présentons tout d'abord le cas des familles d'incompatibilité avant d'introduire les graphes de compatibilité qui permettent de modéliser l'ensemble des contraintes de compatibilité imaginables.

Les familles d'incompatibilité

La notion de famille d'incompatibilité est la notion d'incompatibilité la plus utilisée dans les recherches sur les machines à traitement par batches. Deux tâches de familles différentes ne peuvent appartenir à un même batch, autrement dit une tâche n'est compatible qu'avec des tâches de la même famille. Ce type de compatibilité très simple est transitif, c'est-à-dire que si deux tâches sont compatibles avec une troisième tâche alors elles sont compatibles entre elles. Or cette propriété n'est pas nécessairement vraie. Par exemple pour 3 objets dont les durées de cuisson sont respectivement de 10, 11 et 12 minutes avec possibilité de surcuire les objets une minute supplémentaire, le second objet peut être cuit avec le premier ou (exclusif) avec le troisième objet, mais le premier et le troisième objet ne peuvent être cuits simultanément.

Les graphes de compatibilité

Comme leur nom l'indique les graphes de compatibilité, représentent les compatibilités à l'aide d'un graphe. Ici la compatibilité concerne les tâches entre elles et il n'existe pas de notion de familles de tâches. Deux tâches sont compatibles s'il existe une arête les reliant. Certains auteurs utilisent plutôt le graphe complémentaire, où la présence d'une arête indique une incompatibilité entre les tâches (notamment[69, 73, 178]).

La représentation des compatibilités par les graphes a l'avantage de représenter tout type de contraintes de compatibilité, par exemple les familles d'incompatibilité sont représentées par une clique pour chaque famille, avec aucune arête entre deux sommets de cliques distinctes. Des graphes particuliers sont également étudiés parmi lesquels les graphes scindés, les graphes bipartis et les graphes d'intervalles [22, 23, 24].

Rappelons qu'un *Graphe d'intervalle* $G = (\mathcal{I}, E)$ est un graphe pour lequel l'ensemble de nœuds $\mathcal{I} = \{I_1, \ldots, I_n\}$ peut être assimilé à un ensemble d'intervalles tels que deux sommets $I_1,\ I_2 \in \mathcal{I}$ sont adjacents dans G si et seulement si l'intersection des intervalles correspondants à I_1 et I_2 n'est pas vide (c'est-à-dire $I_1 \cap I_2 \neq \emptyset$).

Dans cette thèse nous étudions des problèmes de machines à traitement par batches avec compatibilité entre les tâches de type graphes d'intervalles, mais pour lesquels les intervalles représentent les durées d'exécution des tâches. C'est-à-dire que chaque tâche possède une durée d'exécution minimale et une durée d'exécution maximale, ainsi deux tâches sont compatibles si elles partagent un intervalle de durées d'exécution. Les intervalles sont dits *uniformes* si l'ordre des points initiaux est identique à l'ordre des points finaux. Un intervalle est dit *proportionnel* si l'ordonnée de son point final est proportionnelle à celle de son point initial.

Ainsi les chapitres 3 et 4 traitent le problème de minimisation de la date de fin d'exécution de l'ordonnancement pour un flowshop hybride à deux étages avec au second étage plusieurs machines à traitement par batches avec compatibilité entre les intervalles de durées d'exécution des tâches, alors que le chapitre 5 traite le problème de minimisation de critères réguliers pour une seule machine à traitement par batches avec compatibilité entre les intervalles proportionnels de durées d'exécution des tâches.

1.4 Application industrielle

Le problème de flowshop hybride avec machines à traitement par batches et compatibilité entre les tâches nous a été inspiré par un problème rencontré dans l'industrie du pneumatique. En effet la construction de pneumatiques pour véhicules et camions subit des demandes et des

volumes de production variables d'une gamme à l'autre et dans le temps. Il est donc impossible de dédier une chaîne de production par gamme, d'où la nécessité d'utiliser un outil d'ordonnancement correct.

Décrivons tout d'abord le processus de fabrication d'un pneumatique afin de comprendre le déroulement des opérations, nous décrirons ensuite l'atelier de production relatif au problème qui nous a inspiré. Finalement la dernière partie fait le lien entre le problème industriel et le problème étudié dans cette thèse.

1.4.1 Processus de fabrication d'un pneu

Le processus de fabrication d'un pneumatique s'articule autour de deux phases principales :

1. Phase préparatoire :
   (a) Fabrication de la gomme à partir des matières premières (caoutchoucs naturels ou synthétiques, huiles, pigments, noir de carbone, antioxydants et autres additifs). Pour rendre la gomme vulcanisable pour la dernière phase du cycle de production du souffre est ajouté. À la suite de cette opération la gomme est dite *productive*.
   (b) Fabrication des différentes couches du pneu (un pneu est composé d'une trentaine de composants issues d'une douzaine de familles telles les bandes de roulement, les flancs, les chapes...) par extrusion puis coupe à longueur, avec pour certains composants ajout de fils d'aciers et de toiles (polyester, nylon, rayonne) par calandrage.
2. Phase de construction :
   (a) Assemblage de tous les éléments d'un pneu (bande de roulement, flancs, ceintures...) sur les machines de *construction*. Suite à cet assemblage on obtient un pneu *vert* ou pneu avant vulcanisation.
   (b) Cuisson du pneu vert dans des presses de vulcanisation. La durée de la vulcanisation varie suivant le pneu vert et la dimension des pneus. Suite à cette vulcanisation à plus de 175°C le pneu acquiert sa forme commerciale. Ces presses ressemblent à des gaufriers, avec deux moules amovibles qui s'ouvrent et se ferment simultanément, ils permettent ainsi de cuire deux pneus simultanément. Dans certains ateliers il est possible de trouver des presses pouvant cuire jusqu'à 8 pneus simultanément.

1.4.2 Description de l'atelier de production

L'atelier de production qui nous intéresse sous-traite à d'autres usines du groupe une grande partie de la fabrication des composants, c'est pourquoi nous ne nous intéressons pas aux deux premières phases du processus de production.

Un pneu ne peut-être cuit avant d'être assemblé, de plus les machines sont spécialisées dans l'exécution d'une famille d'opération, c'est-à-dire, que seules les machines d'assemblage peuvent assembler les composants pour obtenir un pneu vert, et seuls les fours peuvent les cuire. Donc l'atelier de production que nous considérons est de type flowshop hybride, avec au premier étage plusieurs machines d'assemblage en parallèle, et au second étage plusieurs machines de cuisson (presses) en parallèle (figure 1.2).

Décrivons tout d'abord l'étage d'assemblage. Un même pneu vert peut être assemblé suivant plusieurs types de construction mais les machines ne supportent pas tous les types de construction. De plus, la durée d'assemblage d'un type de pneu vert varie d'un type de construction à l'autre, cependant pour une construction donnée la durée d'assemblage d'un type de pneu vert

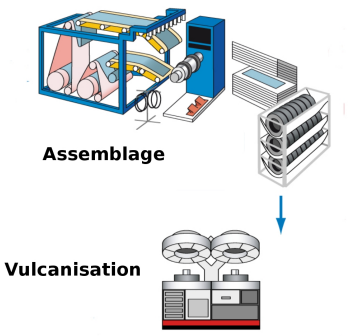


FIGURE 1.2 – Étapes étudiées du système de production d'un pneu

est indépendante de la machine utilisée, à la condition que cette machine supporte le type de construction et la taille du pneu.

L'étage de cuisson est quant-à-lui composé de presses qui sont utilisées pour vulcaniser les pneus. Ces presses sont des machines à traitement par batches capables de vulcaniser deux pneus simultanément avec la même durée de cuisson mais dans des moules amovibles qui peuvent être différents. À la fin de la cuisson les pneus sont commercialisables, or le dessin de la bande de roulement et les inscriptions sur les flancs du pneu sont gravés dans les moules, c'est pourquoi il est impératif de cuire le pneu dans le bon moule. Les différences de dimensions entre les pneus et les différents modes de chauffage des moules impliquent que le moule soit compatible avec la presse. Ces contraintes, auxquelles viennent s'ajouter le nombre réduit de moules, font de la gestion des moules une décision critique du processus d'ordonnancement, non abordée dans cette thèse. De plus chaque pneu a une durée de cuisson minimale et une durée de cuisson maximale dépendante de la durée de cuisson minimale, en effet un pneu peut être légèrement surcuit (environ 4%). Comme les pneus cuits simultanément dans la même presse ont la même durée de cuisson, ils doivent avoir une plage de durées d'exécution possibles commune.

1.4.3 Du problème industriel au problème scientifique

Le problème de flowshop hybride avec machines à traitement par batches et intervalles de durées d'exécution compatibles étudié dans cette thèse est une simplification de ce problème industriel. En effet, nous supposons que toutes les machines d'assemblages et toutes les presses de vulcanisation sont identiques, que le nombre de moules est illimité et que les durées des changements de moules sont négligeables.

L'étage d'assemblage, avec les machines disjonctives, correspond alors au premier étage du flowshop hybride, alors que l'étage de cuisson, avec les machines à traitement par batches et les compatibilités de durées de cuisson, correspond au second étage.

Chapitre 2

Modèles et travaux antérieurs

L'objectif de ce chapitre est de présenter les travaux antérieurs, la première section rappelle les notions permettant de classifier les problèmes et les méthodes de résolution. La section 2.2 présente les problèmes étudiés dans la littérature concernant les machines à traitement par batches (p-batch), alors que la section 2.3 présente les problèmes de flowshops hybrides avec machines classiques. Finalement la section 2.4 présente les problèmes étudiés dans la littérature combinant flowshops et machines à traitement par batches.

Sommaire

2.1 Classification des problèmes et méthodes de résolution 14
2.1.1 Notations des problèmes d'ordonnancement présentés 14
2.1.2 Classification des problèmes d'ordonnancement 14
2.1.3 Classification de méthodes de résolution 18
2.2 Machines à traitement par batch 22
2.2.1 Problèmes sans contraintes de compatitibilité 23
2.2.2 Problèmes avec contraintes de compatibilité 31
2.3 Flowshop Hybride classique 35
2.4 Flowshop et batches 36
2.5 Conclusion du chapitre 39

2.1 Classification des problèmes et méthodes de résolution

Dans cette section, nous présentons les notations que nous utilisons pour définir les différents problèmes, ces notations définies nous détaillons la notation $\alpha|\beta|\gamma$ afin de nommer simplement les problèmes étudiées antérieurement. Finalement nous classifions les principales méthodes de résolution suivant plusieurs critères.

2.1.1 Notations des problèmes d'ordonnancement présentés

Les notations que nous présentons ici sont rappelées dans le glossaire des notations, elles permettent de définir des problèmes d'ordonnancement sur un étage ou un flowshop hybride, avec machines à traitement par batches.

Le problème considéré consiste à ordonnancer n tâches sur m_i machines à l'étage i. Notons que m désigne le nombre maximum de machines sur un étage ($m = \max_i m_i$), donc si un seul étage possède plusieurs machines, on peut alors noter son nombre de machines m.

A. Caractéristiques des machines

La capacité des machines à traitement par batches de l'étage i est notée b_i. Si la capacité est identique pour toutes les machines à traitement par batches nous la notons b.

B. Caractéristiques des tâches

Si la durée d'exécution de la $i^{\text{ème}}$ opération de la tâche d'indice j est imposée elle est notée p_{ij}. Lorsque la durée d'exécution de la $i^{\text{ème}}$ opération de la tâche d'indice j appartient à un intervalle dans ce cas celui-ci est noté $[a_{ij}, b_{ij}]$, si le point final de l'intervalle dépend de son point initial alors l'intervalle est noté $[a_{ij}, (1 + \alpha_i)a_{ij}]$.

La date de disponibilité au premier étage de la tâche d'indice j est notée r_j. Sa date de fin souhaitée est notée d_j, si cette date de fin est impérative elle est notée $\tilde{d_j}$. Dans le cas de machines à traitement par batches de capacité finie, le volume des tâches n'est pas nécessairement unitaire, le volume de la tâche d'indice j sur l'étage i est noté v_{ij}.

C. Critère d'évaluation

Le poids associé à la tâche d'indice j dans la fonction objectif est noté w_j. La date de fin d'exécution de la tâche d'indice j sur le $i^{\text{ème}}$ étage est notée, C_{ij}, sa date de fin d'exécution pour l'ensemble de l'ordonnancement est notée C_j. Le retard algébrique de la tâche d'indice j noté L_j est égal à $C_j - d_j$ alors que le retard noté T_j est égal au maximum entre 0 et L_j. Enfin l'indicateur de retard de la tâche d'indice j est noté U_j, il est égal à 1 si la tâche d'indice j est en retard ($C_j - d_j < 0$), 0 sinon.

2.1.2 Classification des problèmes d'ordonnancement

Dans cette section nous introduisons les notions nécessaires à une classification des problèmes d'ordonnancement, le premier paragraphe permet de différencier les problèmes suivant leurs caractéristiques, et ainsi de résumer un problème suivant un formalisme établi, évitant ainsi de nombreuses lignes d'explication. Le second paragraphe concerne la complexité de résolution de ces problèmes.

A. Classification selon la notation $\alpha|\beta|\gamma$

À l'époque de la normalisation de la notation $\alpha|\beta|\gamma$ les problèmes d'ordonnancement de flowshops hybrides et les problèmes d'ordonnancement de machines à traitement par batches n'étaient pas encore étudiés, c'est pourquoi il a fallu par la suite étendre ces notations. Alors que les notations définies dans Rinnoy Kan [191], Graham et al. [86] et Blazewicz et al. [21], rappelées dans l'ouvrage de Blazewicz et al. [20] publié en 1994, sont utilisées par toute la communauté, pour les machines à traitement par batches nous utilisons les notations de Brucker et al. [28] (1998) et pour les flowshops hybrides celles introduites par Vignier [215] (1997).

Nous décrivons ici la partie de la classification utilisée tout au long de cette thèse. Pour le reste le lecteur peut se référer à Blazewicz et al. [20].

Champ α. Ce champ spécifie les ressources utilisées, leur type, leur nombre et la manière dont elles sont organisées. Nous décomposons ce champ en 3 sous-champs $\alpha = \alpha_1\alpha_2\alpha_3$. Ainsi α_1 décrit le type d'atelier, ensuite α_2 est utilisé uniquement dans le cas de flowshop pour indiquer le nombre d'étages, finalement α_3 décrit le nombre de machines sur chaque étage et leurs relations.

- α_1 décrit la structure de l'atelier.
  - $\emptyset$: un seul étage.
  - F : flowshop.
  - HF : flowshop hybride.
- α_2 décrit le nombre d'étages.
  - s : nombre d'étages égal à s ($s \in \mathbb{N}^*$).
- α_3 décrit le nombre de machines sur chaque étage et leurs relations. S'il n'y a qu'un étage $\alpha_3 = \alpha_4^1\alpha_5^1\alpha_6^1$, s'il y a s étages identiques alors $\alpha_3 = (\alpha_4^i\alpha_5^i\alpha_6^i)$ et si les s étages ne sont pas identiques $\alpha_3 = (\alpha_4^1\alpha_5^1\alpha_6^1, \ldots, \alpha_4^s\alpha_5^s\alpha_6^s)$, enfin si tous les champs α_4^i, α_5^i et α_6^i sont vide alors α_3 l'est aussi,avec $\forall i = 1, \ldots, s$.
  α_4^i décrit le type de machines :
  - $\emptyset$: machine disjonctive.
  - B : machine à traitement par batches de type p-batch.
  - S : machine à traitement par batches de type s-batch.

  α_5^i décrit la relation entre les machines de l'étage.
  - $\emptyset$: une seule machine ou machines identiques en parallèle si absence d'ambiguïté.
  - P : machines identiques en parallèle.
  - R : machines non reliées.

  α_6^i décrit le nombre de machines de l'étage.
  - $\emptyset$: une seule machine, ou aucune information sur le nombre de machines.
  - nb : nombre de machines égal à nb quel que soit l'instance ($nb \in \mathbb{N}^*$).
  - m_i : nombre de machines égal à m_i, varie d'une instance à l'autre.

Champ β. Ce champ permet quant à lui de préciser les caractéristiques des machines, et les contraintes sur les tâches. Nous décomposons ce champs en 11 sous-champs $\beta = \beta_1, \ldots, \beta_{11}$, pour l'ordre de ces sous-champs nous avons préféré mettre en valeur les informations "intéressantes" pour nos problèmes plutôt que de compléter l'ordre habituel. Le premier sous-champ précise le type de compatibilité de ces machines à traitement par batches. Le deuxième sous-champ précise la capacité des machines à traitement par batches. Les sous-champs $\beta_3 \ldots \beta_8$ représentent des contraintes sur le type d'ordonnancement à effectuer (en ligne, attente, permutation, réglages ...), nous ne détaillons pas ces sous-champs ici. Enfin les derniers sous-champs expriment les

contraintes liées aux dates d'arrivée et dates dues ainsi que sur les durées opératoires et les délais de livraison.

- β_1 précise pour les machines à traitement par p-batch le type de compatibilité. La signification des variables de ce champs est expliquée dans la section 2.2.2.
  - $\emptyset$: toutes les tâches sont compatibles (ou absence de machines à traitement par batch).
  - IF : familles incompatibles entre elles.
  - $G = (V; E)$: graphe quelconque.
  - $G = INT$: graphe d'intervalles.
  - $G_p = INT$: graphe d'intervalles de durées d'exécution.
  - $G_p^{\alpha} = INT$: graphe d'intervalles proportionnels de durées d'exécution, c'est-à-dire que les intervalles sont de la forme $[a_j, (1 + \alpha)a_j]$.
  - $G_p^U = INT$: graphe d'intervalles uniformes de durées d'exécution, c'est-à-dire que l'ordre des ordonnées des points initiaux des intervalles est identique à celui des ordonnées des points finaux.
  - $G = \overline{INT}$: complément d'un graphe d'intervalles.
  - $G = (S, K; E)$: un graphe scindé (split graph).
  - $G = (S_1, S_2; E)$: graphe biparti.
  - $G = SPE$: autres graphes spécifiques.
- β_2 précise la capacité des machines à traitement par batches.
  - $b \geqslant n$: La capacité des machines est infinie.
  - $b = k$: toutes les machines sont de capacité finie k, et les tâches de volume unitaire ($k = \{2 \dots n - 1\}$).
  - $b_m = k_m$: les capacités des machines ne sont pas identiques, et les tâches de volume unitaire.
  - $b_s = k_s$: les capacités des machines varient selon les étages, et les tâches de volume unitaire.
  - v_j : le volume des tâches n'est pas unitaire, et la capacité des machines est finie.
  - $s_{jF} = s_F$: le volume des tâches est commun aux tâches de la famille.
  - $v_j, b = k$: le nombre et le volume de tâches sont limités.

Champ γ. Ce dernier champ indique le critère d'optimisation utilisé par le problème. Dans cette thèse nous nous sommes limité aux critères dits *réguliers*, c'est-à-dire que lorsque la date de fin d'exécution d'une tâche est avancée, alors l'ordonnancement résultant est au moins aussi bon. Les critères d'avance-retard sont ainsi exclus de notre travail.

- f_{max} : le maximum d'une fonction régulière des C_j quelconque.
- C_{max} : le makespan (date de fin de traitement de l'ensemble des tâches).
- L_{max} : le retard algébrique maximum (négatif si toutes les tâches se terminent avant leur date due).
- T_{max} : le plus grand retard.
- $\sum f_j$: la somme d'une fonction régulière quelconque.
- $\sum C_j$: la somme des dates de fin de traitement.
- $\sum w_j C_j$: la somme pondérée des dates de fin de traitement.
- $\sum T_j$: la somme des retards.
- $\sum w_j T_j$: la somme pondérée des retards.
- $\sum U_j$: le nombre de tâches en retards.
- $\sum w_j U_j$: la somme pondérée du nombre de tâches en retard.

Notation des problèmes étudiés En appliquant cette notation le problème de minimisation du makespan sur un flowshop hybride à deux étages avec machines à traitement par batches et intervalles de durées d'exécution compatibles est noté $HF2(m_1, Bm_2)|G_p = INT, b = k|C_{max}$, lorsque les durées d'exécution du premier étage sont identiques et les intervalles de durées d'exécution sont uniformes, ce problème est noté $HF2(m_1, Bm_2)|G_p^U = INT, b = k, p_{1j} = p|C_{max}$. Enfin les problèmes de minimisation d'un critère régulier (γ) sur une machine à traitement par batches et intervalles proportionnels de durée d'exécution compatibles sont notés $B|G_p^\alpha = INT, b \geqslant n|\gamma$ et $B|G_p^\alpha = INT, b = k|\gamma$ selon que la capacité des machines à traitement par batches est limitée ou non.

B. Distinction des problèmes selon leur difficulté

En accompagnement de la distinction des problèmes d'ordonnancement suivant les hypothèses, les problèmes sont également discriminés en fonction de leur difficulté de résolution dans le pire cas. Dans un premier temps nous rappelons brièvement les notions de complexité des problèmes de décision, c'est-à-dire les problèmes pour lesquels l'objectif est de répondre par "oui" ou "non", à la question posée sur l'instance fournie. Ensuite nous rappelons le parallèle entre la complexité de ces problèmes de décision et celle des problèmes d'optimisation. Le formalisme présenté ici a été introduit dans l'ouvrage de Garey et Johnson [83].

Un problème appartient à la *classe P*, regroupant les problèmes de décision polynomiaux, s'il est possible de le résoudre en temps polynomial dans le pire cas. C'est-à-dire qu'il peut être résolu par un algorithme de complexité $O(n^k)$ où k est une constante et n désigne la taille de l'instance donnée du problème.

Un problème appartient à la *classe NP*, si toute réponse positive (accompagné de son *certificat*) peut-être vérifiée à l'aide d'un algorithme polynomial. Remarquons que $P \subseteq NP$, en revanche même si la grande majorité de la communauté scientifique pense que l'assertion $P = NP$ est fausse, elle n'a jamais été infirmée. Cependant nous souhaitons utiliser les notions de complexité des problèmes (dans le pire cas) des problèmes afin de pouvoir dire qu'un problème est plus difficile à résoudre qu'un autre, ou que deux problèmes sont équivalents.

Le problème $P1$ est une réduction polynomiale du problème $P2$ s'il existe une fonction polynomiale f qui transforme toute instance du problème $P2$ en une instance du problème $P1$ de telle manière que si la réponse au problème $P2$ est positive pour l'instance d'origine, la réponse au problème $P1$ pour l'instance transformée est également positive, et inversement.

Une classe d'équivalence est une classe dans laquelle tous les problèmes sont des réductions l'un de l'autre. Ainsi la classe P est la plus petite classe d'équivalence des problèmes "faciles" à résoudre en terme de complexité. La classe des problèmes *NP-complets* décrit une autre classe d'équivalence qui contient les problèmes les plus "difficiles" de NP. C'est-à-dire que si un seul problème de cette classe était résolu exactement en temps polynomial alors $P = NP$. À l'opposé si un de ces problèmes était prouvé non résoluble en un temps polynomial alors aucun de ces problèmes ne serait résoluble polynomialement et $P \neq NP$.

Cette catégorie des problèmes NP-complets nécessite d'être affinée, en effet elle regroupe des problèmes NP-complets qui peuvent être résolus par des algorithmes pseudo-polynomiaux (c'est-à-dire polynomiaux en fonction de la valeur maximale des données des instances) alors que pour d'autres problèmes appartenant à NP ceci est impossible. Ainsi alors qu'il existe des algorithmes pseudo-polynomiaux pour résoudre les problèmes NP-complets au sens faible, si $P \neq NP$ il n'existe pas de tels algorithmes pour résoudre les problèmes NP-complets au sens fort. Notons que la classe d'équivalence des problèmes NP-complets au sens fort est basée sur les réductions

pseudo-polynomiales à la place des réductions polynomiales des problèmes NP-complets au sens ordinaire.

En ordonnancement les problèmes utilisés sont généralement des problèmes d'optimisation, ceux de décisions sont relativement rares, c'est pourquoi il est nécessaire d'adapter ces notions aux problèmes d'optimisation. Un problème de décision est associé à un problème d'optimisation par exemple en fixant une borne, la décision est alors "existe-t'il une solution telle que la valeur du critère est inférieure à une valeur donnée ?". Nous avons alors les catégories de problèmes suivantes :

- *Polynomial* : peut être résolu polynomialement (Problème de décision polynomial).
- *NP-difficile* : NP-difficile au sens faible ou au sens fort (Problème de décision NP-complet).
- *Ouvert* : complexité non encore démontrée.

2.1.3 Classification de méthodes de résolution

Il existe de nombreuses méthodes pour résoudre des problèmes d'optimisation, c'est pourquoi il est primordial de définir des critères permettant de lister les avantages et inconvénients des différentes méthodes. L'objectif de cette section est double, en effet nous souhaitons à la fois décrire les principales méthodes de résolution et quelques unes de leurs variantes, tout en introduisant des codes permettant de décrire rapidement les méthodes utilisées dans les travaux antérieurs. Dans un premier temps nous présentons les différentes distinctions, selon la qualité de la solution, la durée d'exécution et la nature de la méthode. Puis nous présentons les principales familles de méthodes et leurs classifications.

A. Classification selon la qualité de la solution obtenue

Résoudre des problèmes NP-difficiles de grande taille nécessite des concessions sur la qualité de la solution. Ainsi nous discriminons les méthodes en fonction de l'écart maximal par rapport à la solution optimale qu'elles garantissent ou non.

Performance non garantie. La qualité de certaines méthodes n'est pas garantie analytiquement, c'est-à-dire que seule une comparaison avec la borne inférieure, suite à l'exécution de la méthode, permet d'estimer l'écart entre la solution obtenue et la solution optimale. Le principal problème de ce type de méthodes est qu'il n'est pas possible de borner cet écart avant l'exécution de l'algorithme, en revanche en pratique elles peuvent fournir des solutions proches de l'optimum pour des problèmes de tailles conséquentes.

Performance garantie. Les méthodes avec performance garantie permettent de garantir analytiquement un certain écart par rapport à la solution optimale. Les méthodes exactes sont des méthodes à performance garantie (avec un écart nul) ces méthodes exactes peuvent être également interrompues lorsque l'écart entre la solution obtenue et la borne inférieure est limité afin d'obtenir des méthodes avec performance garantie. Certaines méthodes approchées de type constructif sont également à performance garantie, quand l'erreur totale entraînée par les différentes approximations est bornée.

B. Classification selon la complexité de l'algorithme, et/ou le contrôle de son arrêt

La classification selon la qualité de la solution fournie par les algorithmes n'est pas suffisante, en effet l'énumération de toutes les solutions réalisables est possible quel que soit le problème

d'optimisation discrète considéré. Or cette méthode garantit d'obtenir une solution optimale, ainsi en considérant uniquement le critère de la qualité de la solution obtenue, aucune méthode n'est plus intéressante que l'énumération de toutes les solutions. Le problème d'une telle énumération est que pour des instances de tailles conséquentes d'un problème NP-difficile au sens fort elles peuvent nécessiter un temps d'exécution de plusieurs siècles sur un processeur de dernière génération.

C'est pourquoi en complément de la classification selon la qualité de la solution obtenue les méthodes sont aussi classifiées selon leur complexité et/ou le contrôle de leur arrêt.

La complexité d'un algorithme consiste à donner l'ordre de grandeur du nombre d'opérations unitaires à effectuer pour résoudre un problème de taille donnée. Généralement la taille d'un problème d'ordonnancement est en $O(n)$, où n est son nombre de tâches. D'où les grandes familles de complexité des algorithmes :

- *Algorithme polynomial*, la complexité de l'algorithme est polynomiale par rapport à la taille de l'instance. Si la taille de l'instance est en $O(n)$ alors la complexité d'un tel algorithme est en $O(n^P)$ où P est un entier commun à toutes les instances du problème.
- *Algorithme pseudo-polynomial*, la complexité de l'algorithme est pseudo-polynomiale par rapport à la taille de l'instance. Si la taille de l'instance est en $O(n)$ alors la complexité d'un tel algorithme est en $O(n^{f(t)})$ où $f(t)$ est une fonction polynomiale d'une caractéristique de l'instance (différente de la taille) cette caractéristique peut être la plus grande valeur, la capacité des batches...
- *Algorithme exponentiel*, la complexité de l'algorithme est exponentielle par rapport à la taille de l'instance. Si la taille de l'instance est en $O(n)$ alors la complexité d'un tel algorithme est en $O(2^n)$.

Un algorithme est dit *à "fin" contrôlée*, si son critère d'arrêt peut ne pas dépendre de la solution obtenue. En général ce critère d'arrêt est utilisé pour des méthodes dont la durée d'exécution risque d'être trop importante. Dans ce cas, après s secondes, ou m itérations l'algorithme est stoppé même s'il n'a pas obtenu une solution optimale, et vérifié l'optimalité de celle-ci.

C. Classification selon le schéma général de l'algorithme

Les méthodes peuvent également être classées selon leur schéma de résolution, certaines méthodes sont constructives, d'autres améliorantes, et enfin les dernières seront mixtes.

Constructives. Une méthode est dite *constructive* si elle complète des solutions partielles jusqu'à obtention d'une solution complète. C'est-à-dire que la méthode consiste à prendre une série de décision non modifiables.

Améliorantes. Une méthode est dite *améliorante* si elle génère des solutions totales puis les modifient afin de les améliorer. C'est-à-dire que la méthode consiste à créer des solutions, puis à améliorer ces solutions.

Mixtes. Une méthode *mixte* combine ces deux méthodes, par exemple en créant intelligemment une solution puis en l'améliorant.

D. Principales méthodes

Nous présentons ici, rapidement, les différentes méthodes de résolution utilisées dans la littératures pour résoudre les problèmes d'optimisation. Nous commençons par présenter les mé-

thodes par construction, les plus simples, avant de présenter les méthodes arborescentes puis les méthodes par exploration itérative de l'espace de solution. Les dernières méthodes présentées sont les méthodes hybrides, combinant plusieurs types de méthodes. Finalement le tableau 2.1, présente les notations de ces méthodes utilisées dans les tableaux présentant les travaux antérieurs.

Les méthodes par construction

Appelées généralement ***Heuristiques***, elles produisent des solutions admissibles en partant d'une solution initiale vide qui est complétée à chaque étape jusqu'à l'obtention d'une solution complète. Aucune décision ne peut-être remise en cause par la suite. L'avantage de ce type de méthode est que l'exécution d'une heuristique est très rapide, et sa mise en place est très simple. En revanche les décisions prises à chaque étape consistent à prendre la meilleure décision pour l'objet ajouté sans tenir compte de ses conséquences sur les décisions restantes. Les heuristiques sont donc des méthodes constructives généralement de complexité polynomiale. En effet, la complexité dépend de l'algorithme qui cherche la meilleure décision locale. Certaines heuristiques possèdent également une garantie de performance.

Les méthodes arborescentes

Les Procédures par Séparation Évaluation (*PSE* ou *B&B : Branch & Bound* en anglais) résolvent optimalement des problèmes NP-difficiles en testant implicitement toutes les solutions réalisables, cette méthode est représentée à l'aide d'un arbre de recherche pour lequel chaque nœud correspond à un sous-problème. La racine de l'arbre correspond au problème d'origine. Il y a ainsi autant d'enfants qu'il existe de valeurs distinctes pour l'objet pour lequel on prend la décision. Cette phase est dite de *séparation*. Dans le but de limiter le nombre de nœuds à explorer, la capacité de chaque nœud à obtenir une solution optimale est testée, soit la phase dite d'*évaluation*. Pour un problème de minimisation, si toutes les bornes inférieures du sous-problème sont strictement inférieures à la meilleure solution obtenue, alors le sous-problème peut fournir une solution optimale. Dans ce cas le nœud correspondant à ce sous-problème est exploré de la même manière que son parent. Dans le cas contraire le sous-problème ne peut fournir une meilleure solution que la meilleure solution déjà obtenue, il est donc inutile d'explorer le nœud correspondant à ce sous-problème. Les sous-problèmes intéressants sont séparés tant qu'ils ne peuvent être résolus optimalement à l'aide d'une méthode polynomiale ou pseudo-polynomiale. Ainsi une PSE classique est une méthode exacte de type constructif dont le temps de calcul est exponentiel.

La programmation linéaire en nombres entiers nécessite de modéliser le problème étudié sous forme de programme linéaire en nombres entiers afin de le résoudre à l'aide d'une PSE en évaluant les nœuds avec l'algorithme du simplexe ou une méthode de points intérieurs pour les gros problèmes, puis en séparant à l'aide de différentes coupes... Il est ainsi possible d'utiliser les outils de résolution de la programmation linéaire, par exemple des solveurs libres comme SYMPHONY ou OpenOpt ou commerciaux comme ILOG CPLEX ou X-Press-MP pour les plus connus. Les méthodes utilisant la programmation linéaire en nombres entiers sont ainsi exactes avec une durée d'exécution exponentielle.

La durée d'exécution d'une PSE étant généralement trop importante pour la dérouler entièrement, des méthodes dérivées ont été développées, la plus simple que nous utilisons dans le

chapitre 3, la *PSE interrompue* stoppe l'exécution de la PSE après un certain temps d'exécution (dans notre cas 10 minutes). La *méthode arborescente à nombres de descendants bornés* limite le nombre de descendants explorés pour chaque nœud, par exemple en explorant seulement les 2 meilleurs descendants de chaque nœud. Enfin le *Beam search* consiste à limiter le nombre de nœuds explorés à chaque niveau de l'arborescence, en prenant par exemple les 50 ou 100 meilleurs nœuds de chaque niveau. Ces trois méthodes ont l'avantage d'être à fin contrôlée, elles perdent cependant leur garantie d'optimalité, hormis la PSE interrompue si elle se termine dans les temps.

La Programmation Dynamique fixe séquentiellement la valeur des différentes inconnues. La caractéristique essentielle, à l'application d'une méthode de programmation dynamique, est que l'on arrive à définir des familles de sous-problèmes de telle sorte que chaque sous-problème, inclus dans un problème plus général, soit résolu optimalement grâce à la résolution optimale de ses propres sous-problèmes. Les méthodes utilisant la programmation dynamique sont des méthodes constructives exactes dont le temps de calcul est en général pseudo-polynomial (voir polynomial), dans certains cas le temps de calcul peut être exponentiel.

Dans le cas où le temps de calcul de l'algorithme de programmation dynamique est exponentiel, il est possible d'effectuer des résolutions approchées des sous-problèmes en limitant, par exemple les valeurs possibles des inconnues, dans ce cas on parle de *programmation dynamique maillée*. L'algorithme de programmation dynamique maillé ne fournit pas une solution exacte, en revanche il peut être polynomial ou pseudo-polynomial, et peut également avoir une garantie de performance.

Les méthodes d'exploration itérative de l'espace des solutions

Le *voisinage* est une notion générique qui regroupe les méthodes explorant le voisinage de solutions admissibles pour y chercher, si possible, de meilleures solutions. Dans les problèmes d'ordonnancement le voisinage d'une solution σ comprend généralement toutes les solutions admissibles atteignables à partir de σ en modifiant légèrement l'ordonnancement. Par exemple l'inversion de deux tâches dans une solution fournit un voisin de la solution d'origine. Ces méthodes sont améliorantes et sont à fin contrôlée. Généralement, elles ne fournissent aucune garantie de performance, et leur complexité n'est pas bornée a priori. Parmi les méthodes de voisinage, nous distinguons les plus connues, le *recuit simulé* et la recherche *tabou*. Dans nos tableaux récapitulatifs, lorsqu'une méthode de voisinage utilise une stratégie différente nous la notons *voisinage*.

Le *recuit simulé* est inspiré d'un algorithme permettant de simuler la stabilisation thermique d'un système physique instable. On génère un voisin, s'il est meilleur alors il est accepté, sinon il est accepté ou rejeté en fonction d'une certaine probabilité qui dépend de la détérioration et d'un paramètre t correspondant à la température. Un schéma de refroidissement précis permet de faire décroître la température, généralement elle est modifiée après un certain nombre d'itérations à température constante. L'algorithme s'arrête lorsqu'aucune solution n'a été acceptée durant un cycle complet d'itérations à température constante.

La recherche *tabou* est une méthode de descente évoluée. Dans sa première version, elle consiste à aller vers le meilleur voisin qui n'a pas été visité durant les T dernières itérations (T peut être suffisamment grand pour contenir toutes les solutions déjà visitées). Les dernières versions interdisent plutôt l'inverse des N dernières modifications utilisées pour modifier la

solution. L'algorithme est arrêté après un certain nombre d'itérations (ou certain temps) sans amélioration de la meilleure solution.

Les approches *évolutionnaires* s'inspirent de phénomènes naturels. Elles consistent à modifier un ensemble de solutions (population) pour qu'il contienne de bonnes solutions. Après avoir construit une population initiale, aléatoirement ou à l'aide de méthodes constructives, la méthode tente d'améliorer la qualité moyenne de la population à l'aide des principes de l'évolution naturelle. Chaque itération (cycle) d'une méthode évolutionnaire est composée d'une phase de coopération durant laquelle les solutions courantes sont comparées puis combinées. Durant la deuxième phase dite d'adaptation individuelle, chaque solution est améliorée indépendamment des autres. La dernière phase est une phase de séelection, qui permet de limiter la taille de la population. À l'instar des méthodes par voisinage il existe plusieurs stratégies présentées dans [152] tels les algorithmes génétiques, les méthodes de *recherche distribuées* (*Scatter Search* en anglais) et l'*algorithme de la fourmi* (ACO). À l'instar des méthodes par voisinage,en général, les approches évolutionnaires ne fournissent aucune garantie de performance et leur complexité est non bornée, d'où la nécessité d'un contrôle de la fin.

Notons que toute méthode itérative fournit la meilleure solution admissible qu'elle a réussi à construire.

Les méthodes hybrides

Les méthodes hybrides combinent les méthodes que nous avons introduites précédemment. Elles consistent par exemple en la création de solutions initiales à l'aide d'une heuristique, cette solution représentant le point de départ d'un algorithme de recuit simulé. Dans les tableaux récapitulatifs que nous présentons par la suite, nous préférons classifier ces méthodes hybrides par la méthode simple dominante.

Les schémas d'approximation polynomiaux (*PTAS : Polynomial Time Approximation Scheme*) fournissent, pour un paramètre $\epsilon > 0$, une solution en temps polynomial par rapport à la taille de l'instance, mais la qualité de la solution dépend de la valeur du paramètre ϵ. C'est-à-dire que la solution obtenue est au plus éloignée d'un facteur ϵ de la valeur optimale. Les techniques utilisées pour obtenir une telle méthode sont diverses et variées [2, 153, 169]. Les schémas d'approximation polynomiaux sont des méthodes hybrides à performance garantie dont la durée d'exécution est polynomiale par rapport à la taille de l'instance. Cependant cette durée d'exécution est dépendante de ϵ, elle peut être exponentielle par rapport à $1/\epsilon$ et donc exploser lorsque ϵ tend vers 0.

2.2 Machines à traitement par batch

Dans cette section nous présentons les travaux antérieurs sur les problèmes de machines à traitement par batches (de type p-batch) avec un étage unique. La première partie traite les problèmes sans contraintes de compatibilités entre les tâches d'un batch. Dans la seconde partie, nous présentons différents types de compatibilité entre les tâches d'un même batch. Pour chaque partie nous commençons par les problèmes à une machine avant d'aborder les problèmes avec machines parallèles.

Méthode	Notation
Heuristique	Heuristique (ou complexité si exacte)
Procédure Par Séparation Évaluation	PSE
Programmation Linéaire en Nombres Entiers	PLNE
Programmation Dynamique	Dynamique (ou PD)
Voisinage	Voisinage
Recuit simulé	Recuit simulé
Recherche tabou	Tabou
Approche évolutionnaire	Évolutionnaire
Schéma d'approximation polynomial	PTAS

TABLE 2.1 – Résumé des notations des méthodes utilisées dans l'état-de-l'art.

2.2.1 Problèmes sans contraintes de compatitibilité

Dans cette section nous différencions les résultats concernant les problèmes d'ordonnancement avec dates de disponibilité, et ceux ne les prenant pas en compte. Ensuite nous présentons les principaux résultats concernant des cas particuliers, et enfin des machines parallèles.

La complexité des problèmes n'étant pas la même en fonction des capacités de la machine à traitement par batches nous les abordons séparément. Ainsi nous commençons par le cas le plus simple, la machine à capacité infinie notée $b \geqslant n$, c'est-à-dire qu'un batch peut contenir toutes les tâches. Ensuite nous traitons le cas des machines avec une capacité finie noté $b = k$, le dernier cas traité en tant que cas particulier est le cas où l'on limite le volume total des tâches avec des tâches dont le volume n'est pas unitaire, ce cas est noté v_j. Limiter le nombre de tâches équivaut à limiter le volume du batch avec des tâches de volumes unitaires.

A. Sans dates de disponibilité (une machine)

Brucker et al. [28] présentent la complexité des problèmes d'ordonnancement d'une machine à traitement par batches pour les critères réguliers, c'est pourquoi nous rappelons ici principalement ces résultats sous forme de tableaux récapitulatifs. À défaut de mention contradictoire, les résultats obtenues dans les tableaux suivant proviennent des travaux de Brucker et al. [28]. La partie de l'état de l'art de Potts et Kovalyov [185] concernant les machines de type p-batch résume ces résultats. Étant donné que Brucker et al. [28] ne présentent pas de problèmes avec volumes de tâches non unitaires, nous résumons les recherches portant sur ces problèmes dans la section 2.2.1.

Le tableau 2.2 présente les résultats de complexité de problèmes d'ordonnancement sur une machine à traitement par batch de type p-batch, avec capacité infinie. C'est-à-dire que $b \geqslant n$ de manière que la machine puisse exécuter toutes les tâches en parallèle.

Le tableau 2.3 présente quant à lui les résultats de complexité sur une machine à traitement par batches à capacité finie($b = k$). C'est-à-dire que $b < n$ ainsi la machine ne peut exécuter toutes les tâches dans un même batch. Afin de comparer le surcroît de complexité entraîné par le traitement par batches nous rappelons la complexité des problèmes avec capacité unitaire, c'est-à-dire le cas des machines discrètes. Notons que les problèmes avec capacité infinie peuvent être moins difficile à résoudre que leurs équivalents à capacité finie.

Fonction objectif	Complexité	Méthode de résolution
f_{max}	Polynomial	
C_{max}	Polynomial	$O(n)$
L_{max}	Polynomial	$O(n^2)$
$\sum_{j=1}^{n} f_j$	NP-difficile au sens faible	Programmation dynamique $O(n^2P)^*$
$\sum_{j=1}^{n} C_j$	Polynomial	$O(n \log n)$
$\sum_{j=1}^{n} w_j C_j$	Polynomial	$O(n \log n)$
$\sum_{j=1}^{n} T_j$	NP-difficile au sens faible [151]	Programmation dynamique $O(n^2P)$
$\sum_{j=1}^{n} w_j T_j$	NP-difficile au sens faible	Programmation dynamique $O(n^2P)$
$\sum_{j=1}^{n} U_j$	Polynomial	$O(n^3)$
$\sum_{j=1}^{n} w_j U_j$	NP-difficile au sens faible	Programmation dynamique $O(n^2P)$

$^*P = \sum_{j=1}^{n} p_j$

TABLE 2.2 – Complexité des problèmes à une machine à traitement par batches, avec capacité infinie, sans date de disponibilité ($B|b \geqslant n|f$).

Fonction objectif	Complexité		Méthode de résolution
	$b = 1$	$b \geqslant 2$	
f_{max}	$O(n^2)$	NP-difficile au sens fort	
C_{max}	$O(n)$	Polynomial	$\min\{O(n \log n), O(n^2/b)\}$
L_{max}	$O(n \log n)$	NP-difficile au sens fort	
$\sum f_j$	NP-difficile au sens fort	NP-difficile au sens fort	
$\sum C_j$	$O(n \log n)$	Ouvert	$O(n^{b(b-1)})$ [28] (PD) $O(h^3 b^{h+1})^*$ [34] (PD) PSE [33, 66] Heuristique [33, 66, 107] PTAS [29]
$\sum w_j C_j$	$O(n \log n)$	Ouvert	PSE [211] Heuristique [211]
$\sum T_j$	$O(n^4P)^{**}$ NP-difficile au sens faible	NP-difficile au sens fort	
$\sum w_j T_j$	NP-difficile au sens fort	NP-difficile au sens fort	
$\sum U_j$	$O(n \log n)$	NP-difficile au sens fort	
$\sum w_j U_j$	$O(nP)$ NP-difficile au sens faible	NP-difficile au sens fort	

* h indique le nombre de durées d'exécution distinctes
$^{**}P = \sum_{j=1}^{n} p_j$

TABLE 2.3 – Complexité des problèmes à une machine à traitement par batches, avec capacité finie, sans date de disponibilité ($B|b = k|f$).

B. Avec dates de disponibilité (une machine)

Contrairement aux problèmes sans dates de disponibilité il n'existe pas d'article étudiant tous les critères réguliers à la manière des travaux de Brucker et al. [28]. Le tableau 2.4 présente les résultats de complexité de problèmes d'ordonnancement sur une machine à traitement par batch de type p-batch, avec capacité infinie ($b \geqslant n$) et dates de disponibilité (r_j).

Type	Complexité	Méthode de résolution	Référence
f_{max}	NP-difficile au sens faible	$O(n^4p^3)^*$	[151]
C_{max}	Polynomial[183]	$O(n \log n)$	[183]
L_{max}	NP-difficile[41]	Heuristique $O(nt(n + t \log n)(1 + \frac{n}{t})^t)^{**}$ PTAS	[150] [41] [17]
$\sum f_j$	NP-difficile au sens faible	$O(n^4p^3)^*$	[151]
$\sum C_j$	Ouvert	PTAS	[58]
$\sum w_jC_j$	NP-difficile au sens faible [58]	PTAS	[60, 147]

$^*p = \max_j\{r_j\} + \sum_j p_j$
$^{**}r_j \in \{r_1, \ldots, r_m\}$, $p_j \in \{p_1, \ldots, p_h\}$, $d_j \in \{d_1, \ldots, d_g\}$: $t = \min\{m, h, g\}$

TABLE 2.4 – Complexité des problèmes à une machine à traitement par batches avec dates de disponibilité et capacité infinie ($B|b \geqslant n, r_j|f$).

Le tableau 2.5 présente quant-à-lui les résultats de complexité de problèmes d'ordonnancement sur une machine à traitement par batch de type p-batch, avec capacité finie ($b = k$) et dates de disponibilité (r_j).

Type	Complexité	Méthode de résolution	Référence
C_{max}	NP-difficile au sens fort	PSE Heuristique PTAS $O(nk(bP_{max}P_{sum})^{k-1})$	[199] [137, 150, 199] [59] [59]
		$O(nc^k(\sum p_j)^{k-1})^*$ Dynamique**	[150] [200]
L_{max}	NP-difficile au sens fort [180]	Heuristique Évolutionnaire	[210] [218, 45]
T_{max}	NP-difficile au sens fort[138]	Heuristique	[89]
$\sum C_j$	NP-difficile au sens fort	PTAS	[57]
$\sum w_jC_j$	NP-difficile au sens fort	PTAS	[147]
$\sum T_j$	NP-difficile au sens fort		[89]
$\sum U_j$	NP-difficile au sens fort [138]	Heuristique	[89]

$^*k$ le nombre de dates de disponibilité distinctes et c une constante
**Familles de même durées

TABLE 2.5 – Complexité des problèmes à une machine à traitement par batches avec dates de disponibilité et capacité finie ($B|b = k, r_j|f$).

C. Cas particuliers (une machine)

Dans cette section nous présentons des cas particuliers de problème d'ordonnancement d'une machine à traitement par batches de capacité finie ou infinie ($b = k$ ou $b \geqslant n$).

Le tableau 2.6 présente les problèmes d'ordonnancement avec une machine à traitement par batches et restrictions sur les durées d'exécution, dates de disponibilité et de dates de fin souhaitées. Le premier problème $(B|b = k, \tilde{d}_j|C_{max})$ introduit des dates de fin impératives pour la minimisation du makespan avec une machine à capacité finie. Ensuite les trois blocs de problèmes limitent le nombre de durées d'exécution ou le nombre de dates de disponibilité différentes. Les derniers problèmes de ce tableau possèdent des durées d'exécution et (ou) dates de disponibilité et (ou) dates de fin souhaitées en bonne concordance.

On dit que les tâches d'un problème sont *en bonne concordance*, ou *accordables* (*agreeable* en anglais) suivant plusieurs critères si trier les tâches suivant un critère les trie également suivant les autres critères. Par exemple si les tâches sont accordables suivant leurs dates de disponibilité et leurs dates de fin souhaitées, les trier par dates de disponibilité croissantes les trie par la même occasion par dates de fin souhaitées croissantes. Dans le cas de tâches accordables, la mention *avec* $x_j \nearrow y_j \nearrow \ldots$ est ajoutée suite à la notation de $\alpha|\beta|\gamma$, où x_j et y_j représentent les critères concordants.

Le tableau 2.7 présente les problèmes d'ordonnancement sur une machine à traitement par batches, sans compatibilité entre les tâches mais avec des contraintes de précédence, des ordonnancements en ligne ainsi que le cas d'intervalles d'indisponibilité de la machine.

Type	Complexité	Méthode	Référence
$B\|b = k, \tilde{d_j}\|C_{max}$	Polynomial	$\min\{O(n \log n), O(n^2/b)\}$	[28]
$B\|b = 2, p_{jF} = p_F\|\sum C_j$	Polynomial	$O(F^2)$	[107]
$B\|b = k, p_j = p\|T_{max}$	Polynomial	$O(n \log n)$	[136]
$B\|b = k, p_j = p\|\sum U_j$	Polynomial	$O(n \log n)$	[136]
$B\|b = k, r_j, p_j = p\|C_{max}$	Polynomial	$O(n)$	[109]
$B\|b = k, r_j, p_j = p\|\sum C_j$	Polynomial	$O(n^3)$	[85, 219]
$B\|b \geqslant n, r_j, p_j = p\|\sum w_j C_j$	Polynomial	$O(n^3)$	[60]
$B\|b \geqslant n, r_j\|\sum w_j C_j$ *avec* $r_j \in \{0, r\}$ *avec* $r_j \in \{r_1, \ldots, r_k\}$ *avec* $p_j \in \{p_1, \ldots, p_k\}$	Polynomial NP-difficile au sens faible NP-difficile	$O(n^2 \log n)$ $O(2^k n^k \ n \log n)$	[60]
$B\|b = k\|T_{max}$ *avec* $p_j \nearrow \ d_j \nearrow$	Polynomial	$O(bn \log(n \max_j\{p_j\}))$	[136]
$B\|b = k\|\sum T_j$ *avec* $p_j \nearrow \ d_j \nearrow$	NP-difficile au sens faible	Dynamique	[148]
$B\|b = k\|\sum w_j T_j$ *avec* $p_j \nearrow \ d_j \nearrow$	NP-difficile au sens fort		[132]
$B\|b = k\|\sum U_j$ *avec* $p_j \nearrow \ d_j \nearrow$	Polynomial	$O(bn^2)$	[136]
$B\|b = k\|\sum w_j U_j$ *avec* $p_j \nearrow \ d_j \nearrow$	NP-difficile au sens faible	Dynamique	[148]
$B\|b \geqslant n, r_j\|L_{max}$ *avec* $p_j \nearrow \ d_j \nearrow$ *avec* $r_j \nearrow \ d_j \nearrow$	NP-difficile Polynomial	$O(n^2)$	[41] [41]
$B\|b = k, r_j\|T_{max}$ *avec* $r_j \nearrow \ d_j \nearrow$ *avec* $r_j \nearrow p_j \nearrow \ d_j \nearrow$	NP-difficile au sens fort Polynomial	$O(n \log \sum p_j)$	[138]
$B\|b = k, r_j\|\sum U_j$ *avec* $r_j \nearrow p_j \nearrow \ d_j \nearrow$ *avec* $r_j \nearrow \ d_j \nearrow$	Polynomial NP-difficile	$O(bn^2)$	[138]
$B\|b = k, r_j\|NT$ *avec* $r_j \nearrow \ d_j \nearrow$	NP-difficile		[138]
$B\|b = k, r_j, p_j = p\|L_{max}$ *avec* $r_j \nearrow \ d_j \nearrow$	Polynomial	$O(n^3)$	[219]
$B\|b = k, r_j, p_j = p\|T_{max}$ *avec* $r_j \nearrow \ d_j \nearrow$	Polynomial	$O(bn \log((n-1)p))$	[136]
$B\|b = k, r_j, p_j = p\|\sum_j U_j$ *avec* $r_j \nearrow \ d_j \nearrow$	Polynomial	$O(bn^2)$	[136]
$B\|b = k, r_j, p_j = p, \tilde{d_j}\|C_{max}$ *avec* $r_j \nearrow \ d_j \nearrow$	Polynomial	$O(n^2)$	[109]

TABLE 2.6 – Complexité des problèmes à une machine à traitement par batches avec restriction sur les durées d'exécution et dates dues.

Type	Complexité	Méthode	Référence
$B\|b \geqslant n, prec, r_j, p_j = 1\|f$	Polynomial	$O(n^2)$	[44]
$B\|b \geqslant n, prec, r_j, p_j = p\|C_{max}$	Polynomial	$O(n^2)$	[44]
$B\|b \geqslant n, prec, r_j, p_j = p\|\sum w_j C_j$	Ouvert	Heuristique	[44]
$B\|b \geqslant n, prec, r_j, p_j = p\|f$	Ouvert	Heuristique	[44]
$B\|b \geqslant n, prec\|f$	NP-difficile au sens fort		[42]
$B\|b \geqslant n, chains\|C_{max}$	NP-difficile au sens fort		[42]
avec $p_j \nearrow prec \nearrow$	Polynomial	$O(n^2)$	[42]
$B\|b \geqslant n, chains\|\sum C_j$	NP-difficile au sens fort		[42]
$B\|b \geqslant n, online\|C_{max}$	NP-difficile		[59, 181, 190, 227]
$B\|b \geqslant n, online, restart\|C_{max}$	NP-difficile		[77]
$B\|b \geqslant n, online, restart_{limit}\|C_{max}$	NP-difficile		[79]
$B\|b = k, online\|C_{max}$	NP-difficile		[182, 227]
$B\|b = k, online, p_j = p\|\sum C_j$	NP-difficile		[75]
$B\|b \geqslant n, online\|\sum w_j C_j$	NP-difficile		[38]
$B\|b = k, online\|\sum w_j C_j$	NP-difficile		[38]
$B\|b \geqslant n, FB\|L_{max} \geqslant 0$	Polynomial	$O(n^2 k_{INT})$ *	[224]
$B\|b \geqslant n, FB\|f_{max}$	Polynomial		[224]
$B\|b \geqslant n, FB\|C_{max}$	Polynomial	$O(n^2 k_{INT})$ *	[224]
$B\|b \geqslant n, FB\|\sum w_j C_j$	Ouvert		[224]
$B\|b \geqslant n, FB\|\sum U_j$	Polynomial	$O(n^4 k_{INT})$ *	[224]
$B\|b \geqslant n, FB\|\sum f_j$	Ouvert	Pseudo-polynomial	[224]

*k_{INT} nombre d'intervalles interdits

TABLE 2.7 – Complexité des problèmes à une machine à traitement par batches (cas particuliers)

D. Volumes variables

Dans le tableau 2.8 nous supposons que la capacité des batches est limitée, mais contrairement à la section précédente le volume des tâches n'est pas unitaire. Ce type de capacité peut être utilisé par exemple lorsque la contenance des batches est limitée par un volume ou un poids total. Ces volumes non-unitaires entraînent un surcroît de complexité non négligeable, ainsi alors que le problème $B|b = k|C_{max}$ est polynomial [28], le même problème avec volumes non-identiques ($B|v_j|C_{max}$) est NP-difficile au sens fort [209]. Ce dernier problème a été l'objet de nombreuses recherches utilisant tous types de méthode de résolution. Hormis les habituels problèmes avec durées d'exécution constantes nous présentons également deux cas particuliers. Le problème $B|v_j, b = k|C_{max}$ est un problème de minimisation du makespan sur une machine à traitement par batches de capacité limités en volume et en nombre de tâches. Le second problème cas particulier, le problème $B|v_j, p_j = \alpha v_j|C_{max}$ permet de traiter des tâches dont la durée d'exécution dépend du volume ou du poids.

Type	Complexité	Méthode	Référence
$B\|v_j\|C_{max}$	NP-difficile au sens fort	Heuristique	[67, 117, 209, 226]
		PSE	[65, 209]
		Recuit simulé	[160]
		Dynamique	[47]
		Évolutionnaire	[39, 119, 231]
		PTAS	[229]
$B\|v_j, b = k|C_{max}$	NP-difficile au sens fort	Recuit simulé	[54]
$B\|v_j, r_j|C_{max}$	NP-difficile au sens fort	Heuristique	[139, 217]
		PTAS	[193]
		Voisinage	[217]
		Évolutionnaire	[48]
$B\|v_j, p_j = p|C_{max}$	NP-difficile au sens fort		[209]
$B\|v_j, p_j = \alpha v_j|C_{max}$	NP-difficile	PTAS	[228]
$B\|v_j, r_j\|\sum F_j$	NP-difficile au sens fort	PLNE	[35]
$B\|v_j\|\sum C_j$	NP-difficile	Heurisitque	[113, 209]
$B\|v_j, r_j\|\sum C_j$	NP-difficile	Heuristique	[37]
$B\|v_j\|\sum w_j C_j$	NP-difficile	PSE	[15]
$B\|v_j, r_j\|\sum w_j T_j$	NP-difficile	Évolutionnaire	[49]

TABLE 2.8 – Complexité des problèmes à une machine à traitement par batches avec volumes des tâches variables (v_j)

E. Machines parallèles

Le tableau 2.9 présente les problèmes de machines parallèles à traitement par batches de caractéristiques présentées précédemment. Les notations sont identiques à celles des problèmes précédents, en revanche lorsque les durées d'exécution sont identiques sur toutes les machines, mais que les capacités varient d'une machine à l'autre, nous considérons que les machines sont identiques (Bm) mais la capacité est noté ($b_m = k_m$).

Type	Complexité	Méthode	Référence
$Bm\|b = k\|\sum C_j$	Ouvert	Heuristique	[33]
$Bm\|b = k, p_{jF} = p_F\|\sum C_j$	Ouvert	$O(F^2 3^m)$ Heuristique	[107]
$Bm\|b = k, r_j\|C_{max}$	NP-difficile au sens fort	Heuristique PTAS	[136] [140]
$Bm\|b = k, r_j\|L_{max}$	NP-difficile au sens fort	Heuristique PTAS	[136] [149]
avec $p_j \nearrow d_j \nearrow$	NP-difficile	Heuristique	[136]
$Bm\|v_j\|C_{max}$	NP-difficile	Évolutionnaire Heuristique Recuit simulé	[52, 118, 128] [51] [36]
$Bm\|b_m = k_m\|C_{max}$	NP-difficile	Évolutionnaire	[192]
$Bm\|b_m = k_m, v_j\|C_{max}$	NP-difficile	Heuristique	[159]
$BR\|b = k\|C_{max}$	NP-difficile au sens fort	Heuristique	[225]
avec $p_{mj} \nearrow\ p_{m'j} \nearrow$	NP-difficile au sens fort	PTAS	[230]
$B2\|b \geqslant n, online\|C_{max}$	NP-difficile		[170]
$Bm\|b \geqslant n, online\|C_{max}$	NP-difficile		[227]
$Bm\|b = k, online, p_j = p\|\sum C_j$	NP-difficile		[76]
$Bm\|v_j, online_{info}, p_{jF} = p_F\|\sum C_j$	NP-difficile		[212]
$Bm\|b = k, online\|\sum w_j C_j$	NP-difficile		[225]

TABLE 2.9 – Complexité des problèmes à plusieurs machines parallèles à traitement par batches sans compatibilité

2.2.2 Problèmes avec contraintes de compatibilité

Après ce résumé des travaux portant sur des problèmes sans contraintes de compatibilité, nous présentons les problèmes d'ordonnancement sur une machine à traitement par batches avec familles incompatibles entre elles étudiés précédemment, puis les problèmes d'ordonnancement sur une machine à traitement par batches avec graphe de compatibilité entre les tâches.

A. Familles incompatibles entre elles (une machine)

Le tableau 2.10 présente les différents travaux réalisés sur les problèmes de minimisation de critères réguliers simples sur une machine à traitement par batches avec familles incompatibles entre elles. Remarquons que la plupart des auteurs supposent que les durées d'exécution des tâches d'une même famille d'incompatibilité sont égales, ce cas est noté $p_{jF} = p_F$.

Type	Complexité	Méthode	Référence
$B\|IF, v_j, p_{jF} = p_F\| \sum C_j$	NP-difficile	Heuristique Évolutionnaire	[63, 64] [125]
$B\|IF, b = k, p_{jF} = p_F\|C_{max}$	Polynomial	$O(n)$	[210]
$B\|IF, b = k, p_{jF} = p_F\|L_{max}$	Polynomial	$O(n \log n)$	[210]
$B\|IF, b = k, p_{jF} = p_F\| \sum w_j C_j$	Polynomial	$O(n \log n)$	[210]
$B\|IF, b = k, p_{jF} = p_F\| \sum T_j$	NP-difficile	Dynamique Heuristique	[161] [130, 161]
$B\|IF, b = k, r_j, p_{jF} = p_F\| \sum T_j$	NP-difficile	Heuristique	[130]
$B\|IF, b = k, p_{jF} = p_F\| \sum w_j T_j$	NP-difficile	Heuristique	[61, 130, 179]
$B\|IF, b = k, r_j, p_{jF} = p_F\| \sum w_j T_j$	NP-difficile	PSE Heuristique	[206] [130, 163]
$B\|IF, b = k, s_{ij}, r_j, p_{jF} = p_F\| \sum w_j T_j$	NP-difficile	Évolutionnaire	[141]
$B\|IF, b = k, p_{jF} = p_F\| \sum U_j$	NP-difficile	Dynamique	[112]
$B\|IF, b = k, p_{jF} = p_F\| \sum w_j U_j$	NP-difficile au sens fort	Dynamique	[143]
$B\|IF, b \geqslant n, r_j\|C_{max}$	NP-difficile au sens fort	Heuristique Dynamique PTAS	[223]
$B\|IF, b = k, r_j\|C_{max}$	NP-difficile au sens fort	PTAS	[171]
$B\|IF, v_j, r_j\|C_{max}$	NP-difficile au sens fort	Heuristique	[171]
$B\|IF, b = k, r_j, p_{jF} = p_F\|C_{max}$	Polynomial	$O(n \log n)$	[210]
$B\|IF, b = k, r_j, p_{jF} = p_F\|L_{max}$	NP-difficile	Heuristique	[210]
$B\|IF, v_j, p_{jF} = p_F\|C_{max}$	NP-difficile	Heuristique	[120]
$B\|IF, v_j, p_{jF} = p_F\| \sum C_j$	NP-difficile	Heuristique	[120]
$B\|IF, v_j, p_{jF} = p_F\| \sum w_j C_j$	NP-difficile	PSE Heuristique Évolutionnaire	[16] [63, 64] [115]
$B\|IF, v_j, p_{jF} = p_F\| \sum w_j T_j$	NP-difficile	PLNE Heuristique Recuit simulé	[72] [72] [71]
$B[IF, b \geqslant n, online\|C_{max}$	NP-difficile		[172]
$B\|2\ IF, b \geqslant n, online\|C_{max}$	NP-difficile		[78]
$B\|IF, s_{jF} = s_F, online, p_j = p\| \sum C_i$	NP-difficile		[213]

TABLE 2.10 – Complexité des problèmes à une machine à traitement par batches avec familles incompatibles entre elles.

B. Graphes de Compatibilité

Le tableau 2.11 présente les différents travaux réalisés sur les problèmes de minimisation de critères réguliers simples sur une machine à traitement par batches avec graphes de compatibilité. Parmi ces problèmes, le critère le plus étudié est le critère de minimisation de la date de fin de l'ordonnancement (makespan), ceci s'explique par le fait que la minimisation du makespan sans dates de disponibilité correspond au problème de coloration pondérée (*weighted coloring*) en théorie des graphes. Notons que pour les problèmes avec graphes d'intervalles, les plus proches de notre problème de durées d'exécution compatibles sont très peu étudiés. Finke et al. [74] ont prouvé, en 2008, la polynomialité du problème $B|G = INT, b \geqslant n|C_{max}$ et indiquent que le problème $B|G = INT, b = k|C_{max}$ est ouvert. Enfin Oulamara et al. [176] ont prouvé que le problème de machine à traitement par batches avec intervalles de durées d'exécution compatibles ($B|G_p = INT, b = k|C_{max}$), étudié dans cette thèse, est résolu à l'aide d'un algorithme polynomial.

Type	Complexité	Méthode	Référence
$B\|G = (V, E), b = 2\|C_{max}$	Polynomial	$O(n^3)$	[22, 121]
$B\|G = (V, E), b = 2, p_j = 1\|C_{max}$	Polynomial	$O(n^{2,5})$	[22]
$B\|G = (V, E), b \geqslant n\|\sum C_j$	NP-difficile	Heuristique	[69]
$B\|G = INT, b \geqslant n\|C_{max}$	Polynomial	$O(n^3)$	[74]
$B\|G = INT, b = k\|C_{max}$	Ouvert		[74]
$B\|G = \overline{INT}, b \geqslant n\|C_{max}$	NP-difficile [73, 178]	Heuristique	[121, 178]
$B\|G_p = INT, b = k\|C_{max}$	Polynomial	$O(n \log n)$	[176]
$B\|G = (S, K; E), b \geqslant n\|C_{max}$	NP-difficile		[56]
$B\|G = (S, K; E), b = k\|C_{max}$	NP-difficile		[25]
$B\|G = (S, K; E), b \geqslant n, p_j = 1\|C_{max}$	Polynomial	$O(n)$	[25]
$B\|G = (S, K; E), b = k, p_j = 1\|C_{max}$	Polynomial	$O(n^3)$	[25]
$B\|G = (S, K; E), b = k, r_j, p_j = 1\|C_{max}$	NP-difficile		[23]
$B\|G = (S, K; E), b = 2, r_j, p_j = 1\|C_{max}$	NP-difficile au sens fort		[23]
$B\|G = (S, K; E), b = k, r_S, r_K = 0, p_i = 1\|C_{max}$	Polynomial	$O(n^3 \log(n/b))$	[23]
$B\|G = (S, K; S \times K), b = k, r_i, p_i = 1\|C_{max}$	Polynomial	$O(n \log n)$	[23]
$B\|G = (S_1, S_2; E), b = 2\|C_{max}$	Polynomial	$O(n^3)$	[24]
$B\|G = (S_1, S_2; E), b = 2, r_i, p_i = 1\|C_{max}$	NP-difficile au sens fort	Heuristique	[24]
$B\|G = (S_1, S_2; S_1 \times S_2), b = 2, r_i, p_i = 1\|C_{max}$	Polynomial	$O(n \log n)$	[24]
$B\|G = (S_1, S_2; E), b = 2, r_{S_1}, r_{S_2} = 0, p_i = 1\|C_{max}$	Polynomial	$O(n^{2,5} log\ n)$	[24]
$B\|G = SPE, b \geqslant n\|C_{max}$	NP-difficile	PTAS Heuristique	[62] [70, 98, 121]
$B\|G = SPE, b \geqslant n\|\sum C_j$	NP-difficile	PTAS	[98]

TABLE 2.11 – Complexité des problèmes à une machine à traitement par batches avec graphe de compatibilité

C. Machines parallèles et compatibilité

Le tableau 2.12 présente les problèmes de machines parallèles à traitement par batches avec contraintes de compatibilité. Notons que très peu d'articles ce sont intéressés aux machines parallèles à traitement par batches avec compatibilité.

Type	Complexité	Méthode	Référence
$Bm\|IF, b \geqslant n, r_j\|C_{max}$	NP-difficile	PTAS	[144]
$Bm\|IF, b = k, r_j\|C_{max}$	NP-difficile	PTAS	[144]
$Bm\|IF, b = k, p_{jF} = p_F\|C_{max}$	NP-difficile	Heuristique	[210]
$Bm\|IF, b_m = k_m, p_{jF} = p_F\|C_{max}$	NP-difficile	Heuristique	[158]
$Bm\|IF, v_j, p_{jF} = p_F\|C_{max}$	NP-difficile	Heuristique Évolutionnaire	[124]
$Bm\|IF, b = k, p_{jF} = p_F\|L_{max}$	NP-difficile	Heuristique	[210]
$Bm\|IF, b = k, r_j, p_{jF} = p_F\|L_{max}$	NP-difficile au sens fort	Heuristique Évolutionnaire	[156]
$Bm\|IF, v_j, p_{jF} = p_F\|\sum C_j$	NP-difficile	Heuristique Évolutionnaire	[124]
$Bm\|IF, b = k, p_{jF} = p_F\|\sum w_j C_j$	NP-difficile	Heuristique	[210]
$Bm\|IF, v_j, p_{jF} = p_F\|\sum w_j C_j$	NP-difficile =	Heuristique Évolutionnaire	[124]
$Bm\|IF, b = k, p_{jF} = p_F\|\sum w_j T_j$	NP-difficile	Heuristique Évolutionnaire Évolutionnaire	[18] [196]
$Bm\|IF, b = k, r_j, p_{jF} = p_F\|\sum w_j T_j$	NP-difficile	Heuristique Évolutionnaire	[163]
$Bm\|IF, b = k, s_{ij}, r_j, p_{jF} = p_F\|\sum w_j T_j$	NP-difficile	Évolutionnaire	[142]

TABLE 2.12 – Complexité des problèmes à plusieurs machines parallèles à traitement par batches avec compatibilité

2.3 Flowshop Hybride classique

Les problèmes de flowshops classiques sont énormément étudiés depuis les travaux de Johnson [111], il y a plus d'un demi-siècle (1954), ainsi en 2006, Gupta et Stafforned [92] ont publié un état-de-l'art des différents travaux qui concernent les problèmes de type flowshop. Cet état-de-l'art étant récent nous ne résumons pas les travaux concernant des problèmes de flowshops classiques.

À l'instar des problèmes de flowshops classiques il existe également deux états-de-l'art de Linn et Zhang [146] et de Vignier, Billaut et Proust [216] pour les problèmes de flowshops hybrides avec machines disjonctives mais ceux-ci datant de 1999, il est nécessaire de résumer les travaux ultérieurs dans le tableau 2.13. Étant donnée la présence d'un état-de-l'art récent (2005) de Kis et Pesch [123] pour la résolution exacte des problèmes $HFS||C_{max}$ et $HFS||\sum C_j$, nous ne citons pas les articles plus anciens traitant de manière exacte ces problèmes.

Type	Complexité	Méthode	Référence
$HFS\|\|C_{max}$	NP-difficile au sens fort	Exacte PSE Bornes Heuristique Recuit simulé Tabou Évolutionnaire	[123]* [106, 167] [100, 214] [27] [110] [166, 173] [7, 68, 184, 221]
$HF2\|\|C_{max}$	NP-difficile au sens fort [90]	PSE Heuristique	[101] [87]
$HF3\|\|C_{max}$	NP-difficile au sens fort	Bornes Heuristique	[195]
$HFS\|r_j, q_j\|C_{max}$	NP-difficile au sens fort	PSE	[165]
$HFS\|\|\sum C_j$	NP-difficile	Exacte PSE Heuristique	[123]* [14] [27]
$HFS\|\|\sum w_j C_j$	NP-difficile	PLNE Tabou	[81, 204] [81]
$HFS\|\|\sum T_j$	NP-difficile	Heuristique	[135]
$HFS\|\|\sum U_j$	NP-difficile	PSE	[46]
$HF2(m_1, 1)\|\|C_{max}$	NP-difficile au sens fort	PSE Heuristique	[91] [90, 91]
$HF2(1, m_2)\|\|\sum T_j$	NP-difficile au sens fort	PSE Heuristique	[134] [93]
$HFS\|r_j\|\sum w_j T_j$	NP-difficile	Heuristique Voisinage	[220]
$HFS\|s_{ijk}\|C_{max}$	NP-difficile	Heuristique Recuit simulé Tabou	[129] [8] [1]
$HFS\|v_{ij}\|C_{max}$	NP-difficile	Évolutionnaire	[114, 174]

TABLE 2.13 – Complexité des problèmes de flowshops hybrides avec machines discrètes

*État-de-l'art

2.4 Flowshop et batches

Dans cette dernière section sur les travaux antérieurs, nous combinons machines à traitement par batches et flowshop, ainsi le tableau 2.14 présente les résultats obtenus pour des flowshop simples avec machines à traitement par batches. Les types de problèmes étudiés sont les mêmes que pour les sections précédentes auxquelles nous ajoutons quelques contraintes liées à la présence de flowshop (attente entre les étages, batches communs aux étages...). Les derniers résultats présentent des problèmes où un étage est composé d'une machine à traitement par batches de type p-batch, et le second d'une machine de type s-batch.

Le tableau 2.15 présente quant-à-lui les problèmes de flowshops hybrides avec machines à traitement par batches. L'essentiel des études combinant ces deux types de problèmes sont des études de processus d'assemblage (noté *assembly*). Les machines du premier étage sont des machines parallèles traitant chacune un composant d'un même produit, ensuite ces composants sont assemblés au second étage. Ainsi le premier étage peut être considéré comme des machines parallèles discrètes indépendantes, alors que le second étage est une machine à traitement par batches avec familles incompatibles entre elles (une famille par produit), le batch devant comporter toutes les tâches (capacité m_1). Il est intéressant de noter que pour les critères de minimisation d'un maximum (makespan et plus grand retard) toutes les tâches d'une même famille appartiennent au même batch lorsque l'on suppose la capacité des batches infinie, par contre pour les critères de minimisation d'une somme il est nécessaire d'imposer une taille de batch. Les applications industrielles de ces problèmes d'assemblage sont nombreuses, par exemple dans la construction d'ordinateurs personnels ([186]), de véhicules d'incendie ([133])...

Type	Complexité	Méthode	Référence
$F2B\|b \geqslant n\|C_{max}$	Polynomial	$O(n \log n)$	[187]
$F2B\|b = k\|C_{max}$	NP-difficile		[187]
$F2B\|v_j\|C_{max}$	NP-difficile	Recuit simulé	[162]
$F2B\|b_I = k, b_{II} \geqslant n\|C_{max}$	NP-difficile		[187]
$F2(1,B)\|b \geqslant n\|C_{max}$	Polynomial	$O(n^2)$	[187]
$F2(B,1)\|b \geqslant n\|C_{max}$	Polynomial	$O(n^3)$	[175]
$F2(B,1)\|b = k\|C_{max}$	NP-difficile	Heuristique	[175]
$F2(1,B)\|G_p = INT, b = k\|C_{max}$	NP-difficile	Heuristique	[176]
$F2(B,1)\|b = k, p_{1j} = p\|C_{max}$	Polynomial	$O(n \log n)$	[3, 175]
$F2(1,B)\|b = k, p_{2j} = p\|C_{max}$	Polynomial	$O(n \log n)$	[3]
$F2B\|b = k, p_{1j} = p, p_{2j} = p\|C_{max}$	Polynomial	$O(n^3)$	[3]
$F3(1,B,1)\|b = k, p_{2j} = p\|C_{max}$	NP-difficile	Heuristique	[3]
$FB\|b_i = k_i, p_{ij} = p\|C_{max}$ $\forall$ étage i	NP-difficile	Heuristique	[202]
$F2(B,1)\|b = k, p_{2j} = p\|C_{max}$	Polynomial	$O(n^2)$	[175]
$F2(B,1)\|b = k, p_{1j} = p\|\sum C_j$	NP-difficile	Heuristique	[3, 108]
$F2(1,B)\|b = k, p_{2j} = p\|\sum C_j$	Polynomial	$O(n^3)$	[3]
$F2B\|b = k, p_{1j} = p, p_{2j} = p\|\sum C_j$	Polynomial	$O(n^3)$	[3]
$FB\|b_i = k_i, p_{ij} = p\|\sum C_j$ $\forall$ étage i	NP-difficile	Heuristique	[202]
$F2(1,B)\|b = k, p_{2j} = p, r_j\|C_{max}$	NP-difficile au sens fort	Heuristique	[201]
$F2B\|v_j, prmu, p_{ij} = p\|C_{max}$	NP-difficile	PLNE	[53]
$F2B\|v_j, prmu\|C_{max}$	NP-difficile	PLNE Heuristique	[53, 145, 116] [145]
$F2B\|IF, b = k, prmu, s_{ijk}\|C_{max}$	NP-difficile	Heuristique	[55]
$F2B\|IF, b = k, prmu, s_{ijk}\|\sum w_j C_j$	NP-difficile	Heuristique	[55]
$F2B\|b \geqslant n, wait_{no}\|C_{max}$	Polynomial	$O(n \log n)$	[177]
$F3B\|b \geqslant n, wait_{no}\|C_{max}$	Polynomial	$O(n^{22})$	[177]
$FB\|b \geqslant n, wait_{no}\|C_{max}$ $r =$ nombre de batches fixé	NP-difficile	$O(n^{m(r-2)+1+\lfloor m/r \rfloor})$	[177]
$F2B\|v_j, prmu, wait_{no}\|C_{max}$	NP-difficile	PLNE Heuristique	[145]
$F2(B,1)\|b = k, wait_{limit}, p_{1j} = p\|C_{max}$	NP-difficile	PLNE Heuristique	[197]
$F2B\|v_j, batch\ commun\|C_{max}$	NP-difficile	Recuit simulé	[128]
$F2(B,S)\|b \geqslant n\|C_{max}$	Polynomial	$O(n^3)$	[175]
$F2(B,S)\|b = k\|C_{max}$	NP-difficile	Heuristique	[175]
$F2(B,S)\|b = k, p_{1j} = p\|C_{max}$	Polynomial	$O(n \log n)$	[3, 175]
$F2(B,S)\|b = k, p_{2j} = p\|C_{max}$	Polynomial	$O(n^2)$	[175]
$F2\|assembly, b = 2\|C_{max}$	NP-difficile		[43]

TABLE 2.14 – Complexité des problèmes combinant flowshop et machines à traitement par batch

Type	Complexité	Méthode	Référence
$HFsRB\|b \geqslant n\|C_{max}$	NP-difficile	Heuristique Évolutionnaire	[12]
$HF2(m_1, B)\|b \geqslant n\|C_{max}$	NP-difficile	Heuristique	[105]
$HF2(m_1, B)\|b \geqslant n, p_2 j = p_2\|C_{max}$	NP-difficile	PTAS	[105]
$HF2(m_1, B)\|b = k\|C_{max}$	NP-difficile	Heuristique	[104]
$HF2(m_1, B)\|2\ IF_{mix}, b \geqslant n, r_j\|C_{max}$	NP-difficile	Heuristique	[122]
$HF2(m_1, 1)\|assembly\|C_{max}$	NP-difficile au sens fort	Heuristique	[126, 186]
$HF2(2, 1)\|assembly\|C_{max}$	NP-difficile au sens fort	Heuristique PSE	[133] [99, 102, 133, 198]
$HF2(m_1, 1)\|assembly, s_{sj}\|C_{max}$	NP-difficile au sens fort	Évolutionnaire	[10]
$HF2(m_1, 1)\|assembly\|L_{max}$	NP-difficile au sens fort	Tabou Évolutionnaire	[9]
$HF2(m_1, 1)\|assembly, s_{ij}\|L_{max}$	NP-difficile	Évolutionnaire	[5]
$HF2(m_1, 1)\|assembly\|\sum C_j$	NP-difficile	Tabou Recuit simulé	[4]
$HF2(m_1, 1)\|assembly, s_{ij}\|\sum C_j$	NP-difficile	Heuristique	[11]
$HF2(m_1, 1)\|assembly\|\sum w_j C_j$	NP-difficile	PSE Heuristique	[208]

TABLE 2.15 – Complexité des problèmes combinant flowshop hybride et machines à traitement par batch

2.5 Conclusion du chapitre

Dans la première partie de ce chapitre, nous avons introduit les différentes notations utilisés dans cette thèse, nous avons également rappelé les notions de base de complexité et de classification des problèmes et méthodes utilisés en ordonnancement.

Dans la seconde partie, consacrée aux problèmes à étage unique avec machines à traitement par batches, nous remarquons que de nombreux travaux étudient les problèmes de machines à traitement par batches sans contraintes de compatibilité. Les problèmes traitant de machines à traitement par batches avec compatibilité entre les tâches sont principalement orientés vers les familles incompatibles entre elles. Finalement, les travaux traitant de graphes de compatibilité sont peu nombreux, et seuls les travaux de Oulamara et al. [176] traitent d'intervalles de durées d'exécution compatibles. Ils présentent un algorithme permettant de résoudre optimalement le problème $B|G_p = INT, b = k|C_{max}$ en temps polynomial.

D'autres types de problèmes avec machines à traitement par p-batch ont également été traités. Ainsi les critères réguliers ne sont pas les seuls à être étudiés, certains articles traitent d'optimisation multicritères (par exemple [6, 103]), d'ordonnancement juste à temps (ex : [127, 164]) mais aussi des ordonnancements cycliques (ex : [88]) ou flous (ex : [40]).

D'autres types d'ateliers ont également été étudiés par exemple les openshops et jobshops (ex : [187]), des problèmes combinant transport et batch (ex : [203])...

Nous avons également décidé de ne pas présenter en détails certaines contraintes liées au batch, par exemple des calcul de durée d'exécution de batches plus élaborés (ex : [95]), des batches semi-continus (ex : [205]), des détériorations (ex : [189]), des délais de livraison (ex : [154, 207, 222]), des pénalités de rejet (ex : [153, 155]), des fenêtres d'exécution (ex : [232])...

Enfin la lecture de l'état-de-l'art de Gupta et Stafforned [92] permet de remarquer que l'important intérêt, de la communauté, pour les problèmes de types flowshops classiques cache un manque d'intérêt pour les problèmes de flowshops hybrides classiques.

Quant-à-la quatrième partie qui combine ces deux grands types de problèmes, elle confirme ces tendances. En effet seul l'article d'Oulamara et al. [176] traite de flowshop avec machines à traitement par batches et compatibilité entre les tâches. En l'occurrence ils étudient le cas particulier du problème de flowshop hybride traité dans les chapitres 3 et 4, où il y a une seule machine sur chaque étage. Aucun article ne traite les problèmes de flowshop hybride avec machines à traitement par batches et compatibilité entre les tâches.

Les problèmes que nous étudions ici, où des machines à traitement par batches avec intervalles de durées d'exécution compatibles uniques sont utilisées dans un flowshop hybride n'ont donc jamais été étudiés.

Chapitre 3

Méthodes approchées

La première section de ce chapitre est consacrée à la résolution approchée du problème de minimisation du makespan d'un flowshop hybride avec machine à traitement par batches et intervalles de durées d'exécution compatibles. Après une courte introduction, la première sous-section présente les schémas génériques des heuristiques proposées. Ensuite nous présentons des bornes inférieures pour nous permettre de mesurer la performance des heuristiques, puis deux heuristiques avec performance garantie pour le problèmes de machines parallèles à traitement par batches et intervalles de durées d'exécution compatibles ($Bm_2|G_p = INT, b = k|C_{max}$). Finalement nous présentons 6 heuristiques et leurs garanties de performance pour le cas général. Heuristiques que nous adaptons également aux cas où il y a une seule machine sur l'un des étages. Dans la dernière partie de cette première section nous présentons les résultats expérimentaux liés à ces heuristiques.

La seconde section présente un schéma d'approximation polynomial (PTAS), pour le cas particulier du problème précédent où les durées d'exécution du premier étage sont identiques, et les intervalles de durées d'exécution du second étage sont uniformes. Après une courte introduction, nous présentons le principe de cette méthode, puis la phase de transformation des données, et deux phases d'ordonnancement de deux groupes de tâches. Finalement nous présentons la méthode en détail.

Sommaire

3.1 Heuristiques 42
3.1.1 Schémas génériques des heuristiques proposées 42
3.1.2 Bornes inférieures 44
3.1.3 Heuristiques pour machines parallèles à traitement par batches 45
3.1.4 Heuristiques pour le flowshop hybride à deux étages 49
3.1.5 Résultats expérimentaux 62
3.2 Durées d'exécution identiques au premier étage 68
3.2.1 Principe 68
3.2.2 Transformation des données 69
3.2.3 Ordonnancement des tâches courtes 72
3.2.4 Ordonnancement des tâches longues 73
3.2.5 Schéma d'approximation polynomial (PTAS) 74
3.3 Conclusion du chapitre 76

3.1 Heuristiques

Dans cette section nous étudions le problème de minimisation du makespan pour un flowshop hybride à deux étages avec machines à traitement par batches et intervalles de durées d'exécution compatibles noté $HF2(m_1, Bm_2)|G_p = INT, b = k|C_{max}$. Or d'après Gupta [90], le problème de minimisation du makespan pour un flowshop hybride à deux étages avec machines disjonctives ($HF2||C_{max}$) est un problème NP-difficile au sens fort. Ce dernier problème est un cas particulier du problème avec machines à traitement par batches. Pour ce cas particulier, la capacité des batches est unitaire, c'est-pourquoi le problème $HF2(m_1, Bm_2)|G_p = INT, b = k|C_{max}$ est NP-difficile au sens fort. L'objectif de cette section est de présenter des méthodes de résolution approchées simples, rapides et avec garanties de performance.

Nous présentons tout d'abord, dans la section 3.1.1 les schémas génériques des heuristiques proposées, puis la section 3.1.2 exhibe les bornes utilisées afin de quantifier la qualité des heuristiques proposées. Avant de présenter les heuristiques avec garantie de performance pour les problèmes de flowshop hybride, nous présentons, dans la section 3.1.3 deux heuristiques avec garantie de performance pour le problème de machines parallèles à traitement par batches avec intervalles de durées d'exécution compatibles ($Bm|G_p = INT, b = k|C_{max}$). Finalement, la section 3.1.4, exhibe 6 heuristiques avec garantie de performance pour le problème $HF2(m_1, Bm_2)|G_p = INT, b = k|C_{max}$, puis ces heuristiques et leur garantie de performances sont adaptées aux cas particuliers contenant une seule machine sur l'un des étages. Finalement la dernière section 3.1.5 présente les résultats obtenus en appliquant ces heuristiques à plusieurs instances.

3.1.1 Schémas génériques des heuristiques proposées

Les différentes heuristiques que nous proposons sont des heuristiques composées de 3 étapes. La première étape crée la liste de batches à ordonnancer sur le second étage, ensuite la deuxième étape trie la liste des tâches à exécuter sur le premier étage, et trie la liste des batches à exécuter sur le second étage. Finalement la troisième étape séquence les tâches sur le premier étage à l'aide de la règle d'ordonnancement de liste et séquence les batches sur le second étage en utilisant la règle d'ordonnancement de liste et la disponibilité des tâches du batch au second étage. La règle d'ordonnancement de liste, ou règle *FAM* (First-Available-Machine), exécute la première tâche de la liste sur la première machine disponible puis supprime cette tâche de la liste. Cette opération est réitérée tant que la liste n'est pas vide.

Comme la minimisation du makespan dans un flowshop hybride est symétrique, c'est à dire que la valeur optimale d'un problème est identique à celle du problème pour lequel l'ordre des étages est inversé, résoudre le problème $HF2(m_1, Bm_2)|G_p = INT, b = k|C_{max}$ équivaut à résoudre le problème $HF2(Bm_2, m_1)|G_p = INT, b = k|C_{max}$, où ce dernier est appelé *problème inverse* par opposition au *problème initial* ($HF2(m_1, Bm_2)|G_p = INT, b = k|C_{max}$). Par exemple, pour l'ordonnancement des tâches du tableau 3.1 suivant la liste de batches $S = \{(T1, T8), (T6, T7), (T3, T5), (T2; T4)\}$, en plaçant les tâches et les batches au plus tôt nous obtenons l'ordonnancement décrit par le diagramme de Gantt 3.1 pour le problème $HF2(m_1, Bm_2)|G_p = INT, b = k|C_{max}$, et celui décrit par le diagramme de Gantt 3.2 pour le problème *inverse* $HF2(Bm_2, m_1)|G_p = INT, b = k|C_{max}$.

Cette symétrie nous a incité à développer les méthodes également pour le problème inverse, ainsi la création de batches s'effectue avec le même type d'algorithme, la liste de batches résultante et la liste de tâches sont triées. Ensuite les batches sont séquencés en respectant la règle d'ordonnancement de liste sur le premier étage, l'étage qui contient les machines à traitement par

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
p_j	4	7	5	6	8	6	10	4
$[a_j; b_j]$	[5,15]	[3,6]	[7,10]	[3,11]	[9,12]	[11,16]	[15,18]	[14,19]

TABLE 3.1 – Durées d'exécution des tâches (inversion des étages).

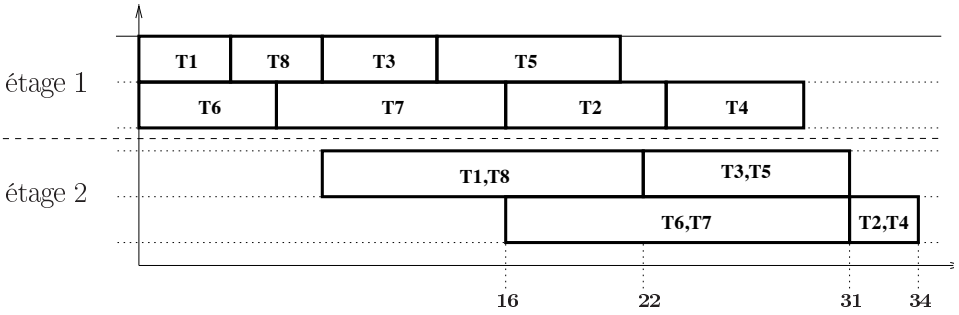


FIGURE 3.1 – Ordonnancement du problème initial (inversion des étages).

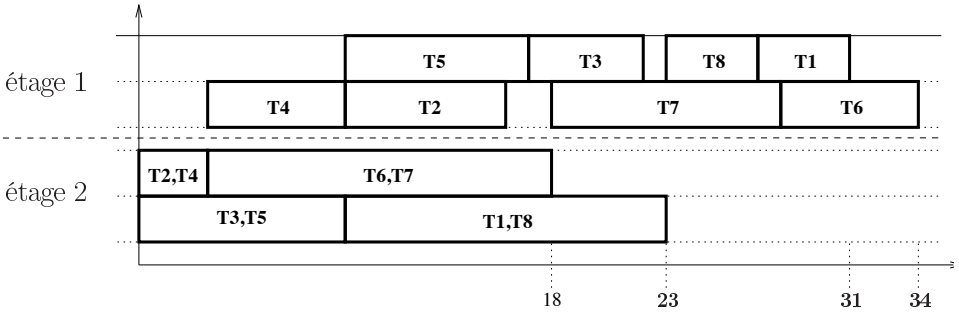


FIGURE 3.2 – Ordonnancement du problème inverse (inversion des étages).

batches. Les tâches sont séquencées quant-à-elles à l'aide de la règle d'ordonnancement de listes en respectant la disponibilité des tâches au second étage. Nous remarquons alors que lorsque nous ne souhaitons pas que l'ordonnancement des tâches respecte un certain ordre, il peut être plus intéressant de ne pas trier les tâches avant d'avoir complété l'ordonnancement du second étage.

En résumé, lorsque les machines à traitement par batches sont au second étage nous obtenons le schéma suivant :

Schéma des heuristiques pour le problème initial

1. Créer les batches en utilisant un algorithme spécifique. Soit L_B la liste des batches créée.
2. Trier la liste des batches L_B suivant la règle définie par l'heuristique H.
3. Trier la liste des tâches L suivant la règle définie par l'heuristique H.
4. Ordonnancer les tâches sur le premier étage suivant la règle FAM.
5. Lorsque toutes les tâches du premier batch de la liste L_B sont disponibles au second étage, exécuter le batch sur la première machine disponible. Supprimer le premier batch de la liste L_B. Réitérer cette opération tant que la liste L_B n'est pas vide.

Pour le problème inversé, c'est-à-dire lorsque les machines à traitement par batches sont au premier étage, et si nous n'imposons aucun ordre sur l'ordonnancement des tâches, nous

obtenons le schéma suivant :

Schéma des heuristiques pour le problème inverse

1. Créer les batches en utilisant un algorithme spécifique. Soit L_B la liste des batches résultants.
2. Trier la liste de batches L_B suivant la règle de l'heuristique H.
3. Ordonnancer les batches sur le premier étage suivant la règle FAM (First-Available-Machine).
4. Si de nouvelles tâches sont disponibles à la date de disponibilité de la première machine, les ajouter à la liste L. Trier la liste L suivant l'ordre défini par l'heuristique H.
5. Si aucune tâche n'est disponible à la date de disponibilité de la première machine, fixer cette date à la date de disponibilité de la première tâche et retourner à l'étape 4, si toutes les tâches ont été ordonnancées arrêter.
6. Ajouter la première tâche sur la première machine et aller à 4.

3.1.2 Bornes inférieures

Les bornes que nous présentons ici, nous permettent de borner l'écart par rapport à l'optimum des heuristiques lorsque nous testons leur efficacité. Chacune des trois bornes proposées traduit une obligation liée à l'ordonnancement. Ainsi l'obligation d'exécuter chaque tâche sur les deux étages implique la nécessité de la première borne. Toutes les tâches doivent également être exécutées sur le premier étage d'où la second borne, enfin toutes les tâches doivent être exécutées sur le second étage d'où la troisième borne.

La première borne (LB_1) traduit la nécessité d'exécuter chaque tâche sur le premier puis sur le second étage.

Soit J l'ensemble des tâches à ordonnancer.

$$LB_1 = \max_{j \in J}\{p_j + a_j\}$$

Toutes les tâches doivent être exécutées sur le premier étage, alors la borne $\frac{\sum_{j \in J} p_j}{m_1}$ est une borne inférieure de la durée d'exécution totale des tâches sur le premier étage. De plus au moins une tâche doit être exécutée sur le second étage après l'instant t, hors la durée d'exécution de toute tâche est supérieure à $\min_{j \in J} a_j$ d'où la borne LB_2 :

$$LB_2 = \frac{\sum_{j \in J} p_j}{m_1} + \min_{j \in J} a_j$$

De la même manière, toutes les tâches doivent être exécutées sur les machines du second étage. En inversant les étages, la date de fin d'exécution des batches sur le premier étage (machines à traitement par batches) est supérieure ou égale à la solution optimale du problème d'ordonnancement de machines parallèles à traitement par batches avec intervalles de compatibilité entre les tâches ($Bm_2|G_p = INT, b = k|C_{max}$). Or le problème d'ordonnancement de machines parallèles disjonctives ($P||C_{max}$) est NP-difficile au sens faible d'après Garey et Johnson [82], donc le problème $Bm_2|G_p = INT, b = k|C_{max}$ est également NP-difficile. C'est pourquoi nous utilisons une borne inférieure de cette date de fin d'exécution. De la même manière que pour la borne précédente, nous utilisons la charge minimale d'une machine à traitement par batch divisée par le nombre de machines. Oulamara et al. [176] ont développé la règle FCBLPT, elle permet de

résoudre optimalement en temps polynomial le problème $B|G_p = INT, b = k|C_{max}$. Cette règle nous permet également de créer les batches lors de la première étape des heuristiques.

Règle FCBLPT

1. Réindexer les tâches par ordre décroissant du point initial de leurs intervalles de durées d'exécution du second étage $a_j,\ j = 1, \ldots, n$. Soit L la liste des tâches, et L_B la liste des batches à ordonnancer, $L_B = \emptyset$ et $i = 0$.
2. À l'itération i, construire un batch B_i contenant la première tâche j de L et les $k-1$ premières tâches de L compatibles avec la tâche j. Supprimer les tâches du batch B_i de la liste L et ajouter le batch B_i à la fin de la liste L_B. Réitérer tant que $L \neq \emptyset$.

Cette règle étant utilisée tout au long de la thèse dans un soucis de compréhension nous présentons ici un exemple d'application de cette règle pour l'instance de 8 tâches présentée dans le tableau 3.2, où la capacité est égale à 3. Les compatibilités entre les tâches sont données par le graphe de compatibilité 3.3. L'ordonnancement fourni par la règle FCBLPT est représenté par le diagramme de Gantt 3.4. Le premier batch contient les tâches $T_7,\ T_8$ et T_6, le second batch contient les tâches $T_5,\ T_3$ et T_1, le dernier les tâches T_2 et T_4.

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
$[a_j; b_j]$	[5,15]	[3,6]	[7,10]	[3,11]	[9,12]	[11,16]	[15,18]	[14,19]

TABLE 3.2 – Durées d'exécution des tâches (FCBLPT).

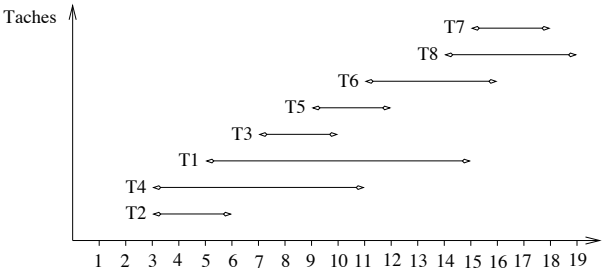


FIGURE 3.3 – Graphe de compatibilité (FCBLPT)

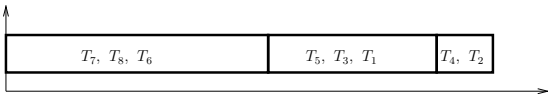


FIGURE 3.4 – Ordonnancement optimal (FCBLPT).

La charge minimale d'une machine à traitement par batches avec intervalles de durées d'exécution compatibles résultant de l'application de la règle FCBLPT est noté $C_{max}^{FCBLPT}(J)$. Ainsi la date de fin d'exécution du dernier batch sur le premier étage (machines à traitement par batch) est supérieure à $\frac{C_{max}^{FCBLPT}(J)}{m_2}$, ce dernier batch est composé au moins d'une tâche qui doit être exécutée sur l'étage de machines disjonctives. La durée d'exécution de cette tâche est supérieure à $\min_{j \in J} p_j$ d'où la borne :

$$LB_3 = \frac{C_{max}^{FCBLPT}(J)}{m_2} + \min_{j \in J} p_j$$

3.1.3 Heuristiques pour machines parallèles à traitement par batches

Avant de présenter les heuristiques pour le flowshop hybride, nous détaillons ici le schéma général et les garanties de performance des heuristiques pour machines parallèles à traitement

par batches. Ces résultats nous seront utiles lors des présentations des différentes heuristiques pour le flowshop hybride.

Cette section consacrée au problème de machines parallèles à traitement par batches présente la garantie de performances de deux heuristiques simples pour les problèmes de minimisation du makespan sur des machines parallèles à traitement par batches avec intervalles de durées d'exécution compatibles ($Bm_2|G_p = INT, b = k|C_{max}$). Ces heuristiques créent les batches en utilisant la règle FCBLPT. Ensuite elles trient la liste de batches obtenus suivant une règle propre à l'heuristique, et enfin elles les séquencent sur les machines en utilisant la règle FAM. Autrement dit le schéma générique de ces heuristiques est le suivant :

Schéma des heuristiques pour machines parallèles

1. Appliquer la règle FCBLPT pour construire la liste de batches L_B. Trier la liste L_B suivant l'ordre défini par l'heuristique H.
2. Placer les batches suivant la règle *FAM* (First-Available-Machine). Cette opération est réitérée tant que la liste n'est pas vide.

La différence entre les deux heuristiques concerne l'ordre des batches. Alors que les batches sont triés dans un ordre quelconque dans l'heuristique $\mathrm{H_{FCBList}}$, la liste L_B est triée selon la règle LPT (Largest-Processing-Time) par l'heuristique $\mathrm{H_{FCBLPT}}$, c'est-à-dire par durée d'exécution décroissante.

Lemme 3.1.1 *L'heuristique $H_{FCBList}$ fournit un ordonnancement du problème $Bm_2|G_p = INT$, $b = k|C_{max}$ en $O(n \log n)$ avec un ratio de garantie de performance égal à $2 - \frac{1}{m_2}$, et cette borne est atteinte.*

Preuve. La première étape de l'heuristique $\mathrm{H_{FCBList}}$ est la création des batches, or d'après Oulamara et al. [176] la règle FCBLPT permet d'obtenir des batches dont la durée totale d'exécution est minimale. La seconde étape de l'algorithme ordonnance les batches suivant un algorithme de liste sur des machines parallèles, or l'ordonnancement de liste fournit des ordonnancements dont le ratio de garantie de performance est égal à $2 - \frac{1}{m_2}$.

L'heuristique $\mathrm{H_{FCBList}}$ fournit donc des solutions à moins de $2 - \frac{1}{m_2}$ fois la solution optimale. Nous exhibons ici une instance du problème pour laquelle l'heuristique fournit une solution égale à $2 - \frac{1}{m_2}$ fois la solution optimale.

Soit I une instance du problème Bm_2, avec m machines, la capacité des batches est $\max\{2, nb_D\}$. Cette instance est composée de $m-1$ tâches de type A, de $m-1$ tâches de type B, d'une tâche de type C et de nb_D tâches de type D. Les durées d'exécution des tâches sont indiquées dans le tableau 3.3, dans lequel $\epsilon < \min\{\frac{m}{3}; \frac{(m-1)^2}{2m+1}\}$. Alors les batches de l'ordonnancement optimal et ceux de l'ordonnancement fourni par l'heuristique sont identiques. Les tâches de types A, B et C sont dans des batches unitaires car elles ne sont pas compatibles 2 à 2, alors que les tâches de type D appartiennent au même batch.

S'il n'y a pas de tâches de type D ($nb_D = 0$), alors la date de fin de l'ordonnancement $C_{max}(S_{H_{List}})$ de l'ordonnancement $S_{H_{List}}$ fournie par l'heuristique $\mathrm{H_{List}}$ pour l'instance I est égal à $C_{max}(S_{H_{List}}) = (2m-1)L - (m-2)\epsilon$ (voir figure 3.5) alors que la date de fin d'un ordonnancement optimal pour l'instance I est égal à $C_{max}(S^*) = mL + \epsilon$ (voir figure 3.6). De plus $C_{max}(S^*)$ est optimal car il atteint la borne inférieure $LB_2 = \frac{\sum_{j \in J} p_j}{m} + \min_{j \in J} a_j$. D'où :

$$\frac{C_{max}(S_{H_{LPT}})}{C_{max}(S^*)} = \frac{(2m-1)L - (m-2)\epsilon}{mL + \epsilon}$$

Type	Nombre de tâches	p
A_1	1	$[(m-1)L;(m-1)L]$
A_2	1	$[(m-1)L-\epsilon;(m-1)L-\epsilon]$
		...
A_{m-2}	1	$[(m-1)L-(m-3)\epsilon;(m-1)L-(m-3)\epsilon]$
A_{m-1}	1	$[(m-1)L-(m-2)\epsilon;(m-1)L-(m-2)\epsilon]$
B_1	1	$[L;L]$
B_2	1	$[L+\epsilon;L+\epsilon]$
		...
B_{m-2}	1	$[L+(m-3)\epsilon;L+(m-3)\epsilon]$
B_{m-1}	1	$[L+(m-2)\epsilon;L+(m-2)\epsilon]$
C	1	$[mL;mL]$
D	nb_D	$[\epsilon;\epsilon]$

TABLE 3.3 – Durées d'exécution des tâches de l'instance I (H_{List} pour Bm_2).

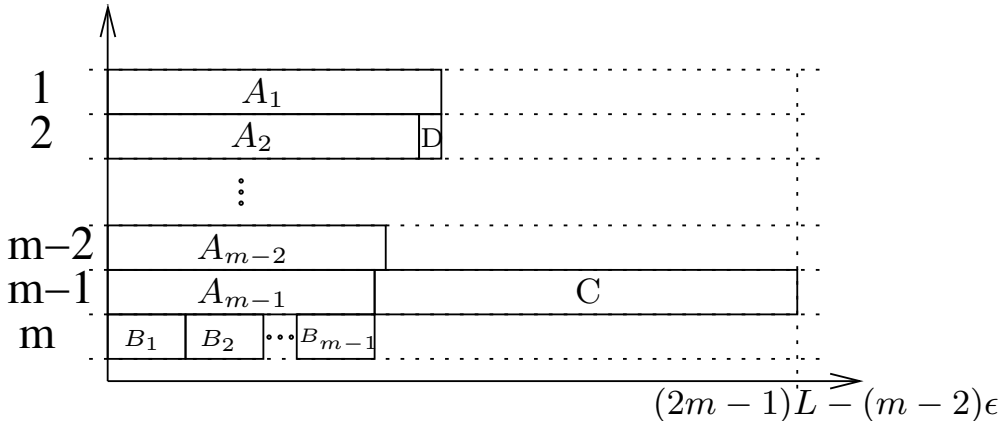


FIGURE 3.5 – Ordonnancement $S_{H_{List}}$ (pour Bm_2).

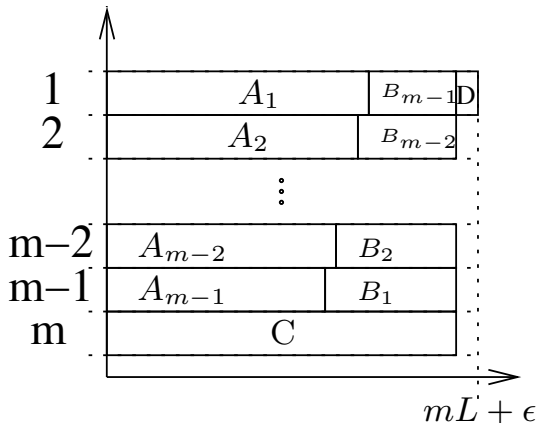


FIGURE 3.6 – Ordonnancement optimal S^* (H_{List} pour Bm_2).

Ainsi :

$$\lim_{\epsilon \to 0} \frac{C_{max}(S_{H_{LPT}})}{C_{max}(S^*)} = 2 - \frac{1}{m}$$

Si $nb_D > 0$, lorsque ϵ tend vers 0 les tâches de type D deviennent insignifiantes.

□

Lemme 3.1.2 *L'heuristique H_{FCBLPT} fournit un ordonnancement du problème $Bm_2|G_p = INT, b = k|C_{max}$ en $O(n \log n)$ avec un ratio de garantie de performance égal à $\frac{4}{3} - \frac{1}{3m_2}$, et cette borne est atteinte.*

Preuve. La première étape de l'heuristique $\mathrm{H_{FCBLPT}}$ est la création des batches, or d'après Oulamara et al. [176] la règle FCBLPT permet d'obtenir des batches dont la durée totale d'exécution est minimale. La seconde étape de l'algorithme est identique à l'heuristique de Lee et al. [136] développée pour le problème de machines parallèles à traitement par batches $Bm_2|b = k|C_{max}$, le ratio de cette heuristique est de $\frac{4}{3} - \frac{1}{3m_2}$.

L'heuristique $\mathrm{H_{FCBLPT}}$ fournit donc des solutions à moins de $\frac{4}{3} - \frac{1}{3m_2}$ fois la solution optimale. Nous exhibons ici une instance du problème pour laquelle l'heuristique fournit une solution égale à $\frac{4}{3} - \frac{1}{3m_2}$ fois la solution optimale.

Soit I une instance du problème Bm_2, avec m machines, la capacité des batches est $\max\{2, nb_c\}$. Cette instance est composée de $2m+1$ tâches de type A et nb_C tâches de type C. Les durées d'exécution des tâches sont indiquées dans le tableau 3.4, dans lequel $\epsilon < \min\{\frac{m}{3}; \frac{(m-1)^2}{2m+1}\}$. Les tâches de type A étant incompatibles 2 à 2 elles appartiennent à des batches unitaires alors que toutes les tâches de type C appartiennent au même batch.

Type	Nombre de tâches	p
A_1	2	$[2m-1; 2m-1]$ et $[2m-1-\epsilon; 2m-1-\epsilon]$
A_2	2	$[2m-2; 2m-2]$ et $[2m-2-\epsilon; 2m-2-\epsilon]$
	...	
A_{m-1}	2	$[m+1; m+1]$ et $[m+1-\epsilon; m+1-\epsilon]$
A_m	3	$[m; m]$, $[m-\epsilon; m-\epsilon]$ et $[m-2\epsilon; m-2\epsilon]$
C	nb_C	$[\epsilon; \epsilon]$

TABLE 3.4 – Durées d'exécution des tâches de l'instance I (H_{LPT} pour Bm_2).

S'il n'y a pas de tâches de type C ($nb_C = 0$), alors la date de fin de l'ordonnancement $C_{max}(S_{H_{LPT}})$ de l'ordonnancement $S_{H_{LPT}}$ fournie par l'heuristique $\mathrm{H_{LPT}}$ pour l'instance I est égal à $C_{max}(S_{H_{LPT}}) = 4m - 1 - 3\epsilon$ (voir figure 3.7) alors que la date de fin d'un ordonnancement optimal pour l'instance I est égal à $C_{max}(S^*) = 3m$ (voir figure 3.8). De plus $C_{max}(S^*)$ est optimal car il atteint la borne inférieure $LB_2 = \frac{\sum_{j \in J} p_j}{m} + \min_{j \in J} a_j$. D'où :

$$\frac{C_{max}(S_{H_{LPT}})}{C_{max}(S^*)} = \frac{4m - 1 - 3\epsilon}{3m}$$

Ainsi :

$$\lim_{\epsilon \to 0} \frac{C_{max}(S_{H_{LPT}})}{C_{max}(S^*)} = \frac{4}{3} - \frac{1}{3m}$$

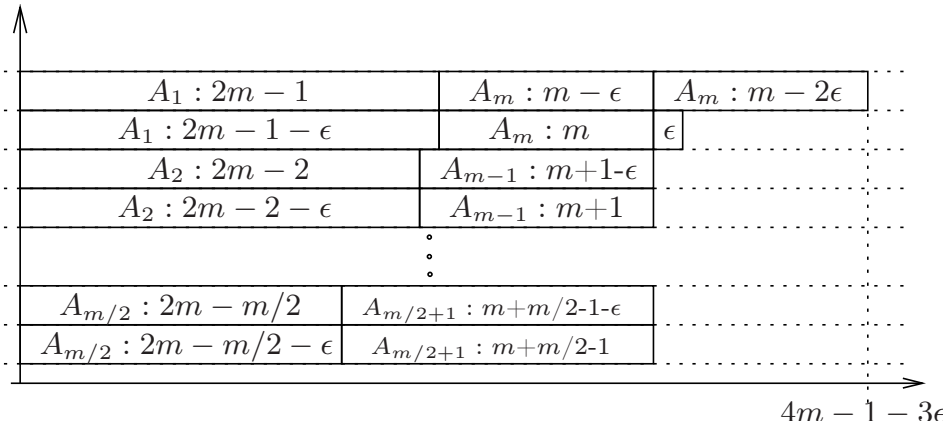


FIGURE 3.7 – Ordonnancement $S_{H_{LPT}}$ (pour Bm_2).

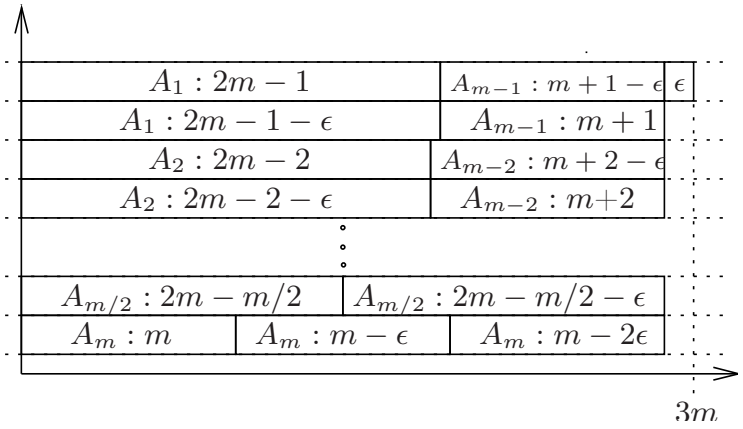


FIGURE 3.8 – Ordonnancement optimal S^* (H_{LPT} pour Bm_2).

Si $nb_C > 0$, lorsque ϵ tend vers 0 les tâches de type C deviennent insignifiantes.

□

3.1.4 Heuristiques pour le flowshop hybride à deux étages

Cette section est décomposée en trois parties, dans la partie A, nous présentons 6 heuristiques pour le cas général, avec plusieurs machines sur chaque étage. Ensuite la partie B, s'intéresse au cas particulier où une seule machine est présente sur le premier étage, nous présentons alors des adaptations des heuristiques générales. La dernière partie (partie C) présente des adaptations des heuristiques générales au cas particulier où une seule machine est présente sur le second étage. Afin de simplifier les notations des problèmes, nous utilisons dans le reste de cette section, uniquement le champ α pour nommer les problèmes. Ainsi le problème initial $HF2(m_1, Bm_2)|G_p = INT, b = k|C_{max}$ est noté $HF2(m_1, Bm_2)$ alors que le problème inverse est noté $HF2(Bm_2, m_1)$.

A. Plusieurs machines à chaque étage

Les heuristiques présentées ici sont développées pour le problème initial $HF2(m_1, Bm_2)$. Les heuristiques sont rassemblées par paires, les premières concernent le problème initial alors que les secondes concernent le problème inverse. Nous avons ainsi développé trois types d'heuristique (6 heuristiques au total). Le premier type d'heuristique, basé sur la règle LPT, trie les tâches et les batches suivant la règle LPT. Ensuite le deuxième type d'heuristique utilise la règle LPT pour trier les batches, mais traite les tâches batch après batch. Finalement le dernier type d'heuristique trie les batches à l'aide de la règle de Johnson et traite les tâches batch après batch.

Par la suite nous détaillons uniquement les résultats relatifs aux heuristiques du problème initial car les heuristiques du problème inverse fournissent les mêmes résultats analytiques.

Heuristique H_{LPT}. Cette heuristique ordonnance les deux étages indépendamment, sur le premier étage les tâches sont ordonnancées suivant les règles LPT et FAM. Au second étage les batches sont créés suivant la règle FCBLPT et triés à l'aide de la règle LPT. Dès que toutes les tâches du premier batch sont disponibles au second étage, le batch est exécuté sur la première machine disponible, et supprimé de la liste. Réitérer cette opération tant qu'il reste des batches dans la liste. Le principal avantage de cette heuristique est de conserver l'ordre LPT sur les deux étages.

Théorème 3.1.3 *L'heuristique H_{LPT} fournit un ordonnancement $S_{H_{LPT}}$ du problème $HF2(m_1, Bm_2)$ en $O(n \log n)$ avec un ratio de garantie de performance de $\frac{8}{3} - \frac{2}{3m}$, et cette borne est atteinte.*

Preuve. Remarquons tout d'abord que la complexité de l'algorithme est donnée par les trois premières étapes qui sont des opérations de tri, d'où la complexité de l'algorithme en $O(n \log n)$. Soit $S_{H_{LPT}}$ l'ordonnancement obtenu avec l'heuristique H_{LPT}, nous considérons les deux étages indépendamment. Au premier étage la règle LPT fournit un ordonnancement avec un ratio de performance garantie égal à $\frac{4}{3} - \frac{1}{3m_1}$. Au second étage, le problème d'ordonnancement de machines parallèles à traitement par batches $(Bm|G_p = INT, b = k|C_{max})$ est résolu approximativement avec un ratio de $\frac{4}{3} - \frac{1}{3m_2}$ par l'heuristique $\mathrm{H_{FCBLPT}}$ (Lemme 3.1.2). D'où :

$$\begin{aligned} C_{max}(S_{H_{LPT}}) &\leqslant C_{max}(S_{H_{Pm_1}}) + C_{max}(S_{H_{Bm_2}}) \\ &\leqslant \left(\tfrac{4}{3} - \tfrac{1}{3m_1}\right) C_{max}(S^*_{Pm_1}) + \left(\tfrac{4}{3} - \tfrac{1}{3m_2}\right) C_{max}(S^*_{Bm_2}) \end{aligned}$$

Soit S^* l'ordonnancement optimal du problème de flowshop hybride, et $C_{max}(S^*)$ sa date de fin, alors

$$C_{max}(S^*) \geqslant C_{max}(S^*_{Pm_1}) \qquad \text{et} \qquad C_{max}(S^*) \geqslant C_{max}(S^*_{Bm_2})$$

Donc :

$$\begin{aligned} C_{max}(S_{H_{LPT}}) &\leqslant \left(\tfrac{4}{3} - \tfrac{1}{3m_1}\right) C_{max}(S^*_{Pm_1}) + \left(\tfrac{4}{3} - \tfrac{1}{3m_2}\right) C_{max}(S^*_{Bm_2}) \\ &\leqslant \left(\tfrac{8}{3} - \tfrac{1}{3m_1} - \tfrac{1}{3m_2}\right) C_{max}(S^*) \end{aligned}$$

C'est-à-dire : [1]

$$\frac{C_{max}(S_{H_{LPT}})}{C_{max}(S^*)} \leqslant \frac{8}{3} - \frac{2}{3m}$$

1. rappelons que $m = \max\{m_1, m_2\}$

Ainsi l'heuristique H_{LPT} fournit une solution avec un facteur de performance borné par $\frac{8}{3} - \frac{2}{3m}$. Nous présentons une instance pour laquelle cette borne est atteinte.

Soit I une instance du problème de flowshop hybride $HF2(m_1, Bm_2)$, avec m machines sur chaque étage, la capacité des batches est $2m+4$. Cette instance est composée de $2m+1$ tâches de type A, $2m+1$ tâches de type B et nb_C tâches de type C. Les durées d'exécution des tâches sont indiquées dans le tableau 3.5, dans lequel $\epsilon < \min\{\frac{m}{3}; \frac{(m-1)^2}{2m+1}\}$. Dans l'ordonnancement optimal comme dans celui fourni par l'heuristique, la plus petite tâche de type B_1 et les tâches B_2 à B_m appartiennent à des batches unitaires, car elles sont incompatibles entre elles. En revanche alors que l'heuristique rassemble les batches de type A et la plus grande tâche B_1, l'ordonnancement optimal place la plus grande tâche B_1 dans un batch unitaire et les tâches de type A dans un batch. Dans les deux cas les tâches de type C sont regroupées dans un même batch.

Type	Nombre de tâches	p_1	p_2
A_1	2	$2m-1$	$[\epsilon; 2m-1]$
A_2	2	$2m-2$	$[\epsilon; 2m-1]$
		...	
A_{m-1}	2	$m+1$	$[\epsilon; 2m-1]$
A_m	3	m	$[\epsilon; 2m-1]$
B_1	2	ϵ	$[2m-1; 2m-1]$ et $[2m-1-\epsilon; 2m-1-\epsilon]$
B_2	2	ϵ	$[2m-2; 2m-2]$ et $[2m-2-\epsilon; 2m-2-\epsilon]$
		...	
B_{m-1}	2	ϵ	$[m+1; m+1]$ et $[m+1-\epsilon; m+1-\epsilon]$
B_m	3	ϵ	$[m; m]$, $[m-\epsilon; m-\epsilon]$ et $[m-2\epsilon; m-2\epsilon]$
C	nb_C	ϵ	$[\epsilon; \epsilon]$

TABLE 3.5 – Durées d'exécution des tâches de l'instance I (H_{LPT} pour $HF2(m_1, Bm_2)$).

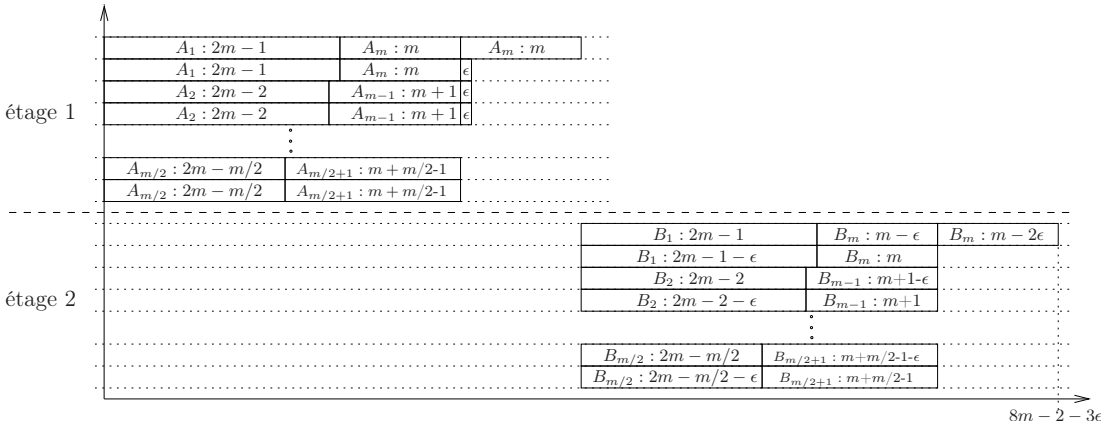


FIGURE 3.9 – Ordonnancement $S_{H_{LPT}}$ (pour $HF2(m_1, Bm_2)$).

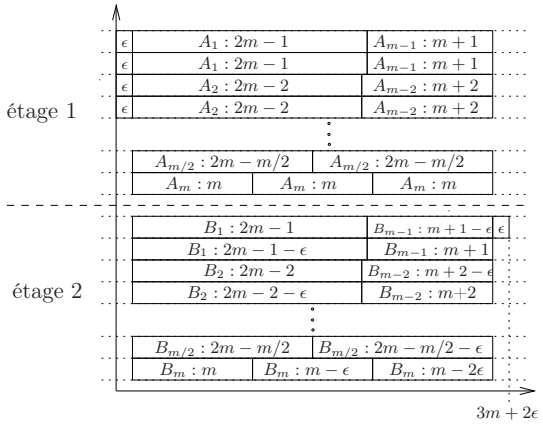


FIGURE 3.10 – Ordonnancement optimal S^* (H_{LPT} pour $HF2(m_1, Bm_2)$).

S'il n'y a pas de tâches de type C ($nb_C = 0$), alors la date de fin de l'ordonnancement $C_{max}(S_{H_{LPT}})$ de l'ordonnancement $S_{H_{LPT}}$ fournie par l'heuristique $\mathrm{H_{LPT}}$ pour l'instance I est égal à $C_{max}(S_{H_{LPT}}) = 8m-2-3\epsilon$ (voir figure 3.9) alors que la date de fin d'un ordonnancement optimal pour l'instance I est égal à $C_{max}(S^*) = 3m + 2\epsilon$ (voir figure 3.10). De plus $C_{max}(S^*)$ est optimal car il atteint la borne inférieure $LB_2 = \frac{\sum_{j\in J} p_j}{m} + \min_{j\in J} a_j$. D'où :

$$\frac{C_{max}(S_{H_{LPT}})}{C_{max}(S^*)} = \frac{8m-2-3\epsilon}{3m+2\epsilon}$$

Ainsi :

$$\lim_{\epsilon\to 0} \frac{C_{max}(S_{H_{LPT}})}{C_{max}(S^*)} = \frac{8}{3} - \frac{2}{3m}$$

Si $nb_C > 0$, lorsque ϵ tend vers 0 les tâches de type C deviennent insignifiantes.
□

La méthode inverse associée à cet algorithme $\mathrm{H_{LPT}}$ n'utilise pas le schéma que nous avons défini pour les heuristiques inverse dans la section 3.1.1, en effet l'ordonnancement de l'étage composé des machines disjonctives doit respecter l'ordre LPT. Il suffit donc d'échanger tâches et batches dans les 2 dernières étapes de l'algorithme $\mathrm{H_{LPT}}$. D'où :

Algorithme H^{Inv}_{LPT}

1. Au deuxième étage, créer les batches en utilisant l'algorithme FCBLPT. Soit L_B la liste des batches créée.
2. Trier la liste des batches L_B suivant la règle LPT.
3. Au premier étage, trier la liste des tâches L suivant la règle LPT.
4. Ordonnancer les batches de la liste L_B sur le premier étage suivant la règle FAM.
5. Lorsque la première tâche de la liste L est disponible au second étage l'exécuter sur la première machine disponible. La supprimer de la liste L. Réitérer cette opération tant que la liste L n'est pas vide.

Théorème 3.1.4 *L'heuristique H^{Inv}_{LPT} fournit un ordonnancement $S^{Inv}_{H_{LPT}}$ du problème $HF2$ (Bm_2, m_1) en $O(n \log n)$ avec un ratio de garantie de performance de $\frac{8}{3} - \frac{2}{3m}$, et cette borne est atteinte.*

Preuve. La preuve est équivalente à celle de $\mathrm{H_{LPT}}$, l'instance pour laquelle $\mathrm{H^{Inv}_{LPT}}$ fournit une solution égale à $\frac{8}{3} - \frac{2}{3m}$ fois l'optimum est également très proche de celle fournie pour $\mathrm{H_{LPT}}$. Afin de ne pas surcharger ce paragraphe, nous ne développons pas cette preuve. □

Heuristique H_{LBPT} (Largest-Batch-Processing-Time). Les règles utilisées par cette heuristique sont identiques à celles utilisées pour l'heuristique $\mathrm{H_{LPT}}$, seulement ici les tâches sont exécutées sur le premier étage en utilisant leur regroupement au sein de batches du second étage. C'est-à-dire qu'elles sont exécutées batch après batch. Au second étage, les batches sont créés suivant la règle FCBLPT et triés à l'aide de la règle LPT. Les tâches du premier batch sont exécutées sur le premier étage en respectant les règles FAM et LPT. Dès que toutes les tâches du premier batch sont disponibles au second étage, le batch est exécuté sur la première machine disponible, puis il est supprimé de la liste. Réitérer ces opérations tant qu'il reste des batches. Cette heuristique fournit en pratique de meilleurs résultats que l'heuristique $\mathrm{H_{LPT}}$ alors qu'étant donné que les tâches du premier étage ne sont plus ordonnancées suivant la règle LPT, la garantie de performance de cette heuristique est moins bonne.

Théorème 3.1.5 *L'heuristique H_{LBPT} fournit un ordonnancement $S_{H_{LBPT}}$ du problème $HF2$ (m_1, Bm_2) en $O(n \log n)$ avec un ratio de garantie de performance de $\frac{10}{3} - \frac{4}{3m}$.*

Preuve. De même que pour l'heuristique précédente, la complexité de l'algorithme est donnée par les trois premières étapes qui sont des opérations de tri, d'où la complexité de l'algorithme en $O(n \log n)$. Soit $S_{H_{LBPT}}$ l'ordonnancement obtenu avec l'heuristique H_{LBPT}, nous considérons les deux étages indépendamment. Le second étage est un problème d'ordonnancement de machines parallèles à traitement par batches $(Bm|G_p = INT, b = k|C_{max})$ résolu approximativement avec un ratio de $\frac{4}{3} - \frac{1}{3m_2}$ par l'heuristique $\mathrm{H_{FCBLPT}}$ (Lemme 3.1.2). Sur le premier étage, le traitement des tâches batches après batch ne permet plus d'obtenir un ordonnancement respectant la règle LPT, cependant nous conservons un ordonnancement de liste. Or un ordonnancement de liste fournit des ordonnancements pour le problème de machines classiques parallèles $(Pm_1||C_{max})$ avec un ratio de garantie de performance de $2 - \frac{1}{m_1}$. D'où :

$$\begin{aligned} C_{max}(S_{H_{LBPT}}) &\leqslant C_{max}(S_{H_{Pm_1}}) + C_{max}(S_{H_{Bm_2}}) \\ &\leqslant \left(2 - \tfrac{1}{m_1}\right) C_{max}(S^*_{Pm_1}) + \left(\tfrac{4}{3} - \tfrac{1}{3m_2}\right) C_{max}(S^*_{Bm_2}) \end{aligned}$$

Soit S^* l'ordonnancement optimal du problème de flowshop hybride, et $C_{max}(S^*)$ sa date de fin, alors

$$C_{max}(S^*) \geqslant C_{max}(S^*_{Pm_1}) \qquad \text{et} \qquad C_{max}(S^*) \geqslant C_{max}(S^*_{Bm_2})$$

Donc :

$$\begin{aligned} C_{max}(S_{H_{LBPT}}) &\leqslant \left(2 - \tfrac{1}{m_1}\right) C_{max}(S^*_{Pm_1}) + \left(\tfrac{4}{3} - \tfrac{1}{3m_2}\right) C_{max}(S^*_{Bm_2}) \\ &\leqslant \left(\tfrac{10}{3} - \tfrac{1}{m_1} - \tfrac{1}{3m_2}\right) C_{max}(S^*) \end{aligned}$$

C'est-à-dire [2] :

$$\frac{C_{max}(S_{H_{LBPT}})}{C_{max}(S^*)} \leqslant \frac{10}{3} - \frac{4}{3m}$$

L'existence d'une instance pour laquelle cette borne est atteinte est une question ouverte.
□

L'heuristique inverse $\mathrm{H}^{\mathrm{Inv}}_{\mathrm{LBPT}}$ respecte le schéma général des heuristiques inverses, présenté dans la section 3.1.1. Elle utilise la règle LPT pour trier la liste de batches.

Théorème 3.1.6 *L'heuristique H^{Inv}_{LBPT} fournit un ordonnancement $S^{Inv}_{H_{LBPT}}$ du problème $HF2$ (Bm_2, m_1) en $O(n \log n)$ avec un ratio de garantie de performance de $\frac{10}{3} - \frac{4}{3m}$.*

Preuve. La preuve est identique à celle de l'heuristique $\mathrm{H}_{\mathrm{LBPT}}$. □

Heuristique H_J (Johnson). Le déroulement de cette heuristique est identique à celui de l'heuristique $\mathrm{H}_{\mathrm{LBPT}}$, seule la règle de tri des batches est modifiée, et donc également la règle de tri des tâches au premier étage. Les batches sont créés suivant la règle FCBLPT et triés à l'aide de la règle de Johnson [111], qui partitionne les tâches en deux groupes : le premier groupe contient les tâches plus courtes sur le premier étage, le second contient les tâches plus courtes sur le second étage. Les tâches du premier groupe sont placées en début de liste par durée d'exécution du premier étage croissante, alors que les tâches du second groupe sont ajoutées en fin de liste par durée d'exécution du second étage décroissantes. Pour notre problème nous remplaçons les tâches par des batches. La durée d'exécution du batch sur le premier étage est égale à la somme des durées d'exécution des tâches du batch divisée par le nombre de machines du premier étage ($\frac{\sum j \in B p_j}{m_1}$) alors que sa durée sur le second étage est égale à la durée d'exécution du batch divisée par le nombre de machines du second étage ($\frac{P(B)}{m_2}$). Les tâches du premier batch sont exécutées sur le premier étage en respectant les règles FAM et LPT. Dès que toutes les tâches du premier batch sont disponibles au second étage, le batch est exécuté sur la première machine disponible, puis le premier batch est supprimé de la liste. Réitérer ces opérations tant qu'il reste des batches.

Théorème 3.1.7 *L'heuristique H_J fournit un ordonnancement S_{H_J} du problème $HF2(m_1, Bm_2)$ en $O(n \log n)$ avec un ratio de garantie de performance de $4 - \frac{2}{m}$.*

Preuve. De même que pour les heuristiques précédentes, la complexité de l'algorithme est donnée par les trois premières étapes qui sont des opérations de tri, d'où la complexité de l'algorithme en $O(n \log n)$. Soit S_{H_J} l'ordonnancement obtenu avec l'heuristique H_J, nous considérons les deux étages indépendamment. L'ordonnancement du second étage ne respecte plus la règle LPT, mais c'est un ordonnancement de liste, or le problème d'ordonnancement de machines parallèles à traitement par batches $(Bm|G_p = INT, b = k|C_{max})$ est résolu approximativement avec un ratio de $2 - \frac{1}{m_2}$ par l'heuristique $\mathrm{H}_{\mathrm{FCBList}}$ (Lemme 3.1.1). Sur le premier étage, l'ordonnancement de liste fournit des ordonnancements pour le problème de machines classiques parallèles $(Pm_1||C_{max})$ avec un ratio de garantie de performance de $2 - \frac{1}{m_1}$. D'où :

$$\begin{aligned} C_{max}(S_{H_J}) &\leqslant C_{max}(S_{H_{Pm_1}}) + C_{max}(S_{H_{Bm_2}}) \\ &\leqslant \left(2 - \tfrac{1}{m_1}\right) C_{max}(S^*_{Pm_1}) + \left(2 - \tfrac{1}{m_2}\right) C_{max}(S^*_{Bm_2}) \end{aligned}$$

2. rappelons que $m = \max\{m_1, m_2\}$

Soit S^* l'ordonnancement optimal du problème de flowshop hybride, et $C_{max}(S^*)$ sa date de fin, alors

$$C_{max}(S^*) \geqslant C_{max}(S^*_{Pm_1}) \qquad \text{et} \qquad C_{max}(S^*) \geqslant C_{max}(S^*_{Bm_2})$$

Donc :

$$\begin{aligned} C_{max}(S_{H_J}) &\leqslant \left(2 - \tfrac{1}{m_1}\right) C_{max}(S^*_{Pm_1}) + \left(2 - \tfrac{1}{m_2}\right) C_{max}(S^*_{Bm_2}) \\ &\leqslant \left(4 - \tfrac{1}{m_1} - \tfrac{1}{m_2}\right) C_{max}(S^*) \end{aligned}$$

C'est-à-dire [3] :

$$\frac{C_{max}(S_{H_J})}{C_{max}(S^*)} \leqslant 4 - \frac{2}{m}$$

L'existence d'une instance pour laquelle cette borne est atteinte est une question ouverte.
□

L'heuristique inverse $\mathrm{H}^{\mathrm{Inv}}_{\mathrm{J}}$ respecte le schéma général des heuristiques inverses, présenté dans la section 3.1.1, en utilisant la règle de Johnson pour trier la liste de batches.

Théorème 3.1.8 *L'heuristique H^{Inv}_J fournit un ordonnancement $S_{H^{Inv}_J}$ du problème $HF2(Bm_2, m_1)$ en $O(n \log n)$ avec un ratio de garantie de performance de $4 - \frac{2}{m}$.*

Preuve. La preuve est identique à celle de l'heuristique $\mathrm{H_J}$. □

B. Une machine au premier étage

Nous traitons désormais le cas où une seule machine est présente sur le premier étage, ce problème est noté $HF2(1, Bm)$. À l'instar du problème avec plusieurs machines sur chaque étage, nous regroupons les heuristiques par paire, la première heuristique de chaque paire est développée pour le problème initial $HF2(1, Bm)$ alors que la seconde est développée pour le problème inverse $HF2(Bm, 1)$. Le premier type d'heuristique, est une adaptation des heuristiques $\mathrm{H_{LBPT}}$ du problème général, ainsi les batches sont triés suivant la règle LPT puis les tâches sont séquencées batch après batch. Le second type d'heuristique, adapté des heuristiques $\mathrm{H_J}$ du problème général qui trie les batches suivant la règle de Johnson puis séquence les tâches batch après batch.

Heuristique H_{LBPT} (Largest-Batch-Processing-Time). Au second étage, les batches sont créés suivant la règle FCBLPT et triés à l'aide de la règle LPT. Les tâches du premier batch sont exécutées sans temps mort sur la machine du premier étage, comme il n'y a qu'une machine l'ordre entre les tâches d'un même batch n'a aucune importance. Dès que toutes les tâches du premier batch sont disponibles au second étage, le batch est exécuté sur la première machine disponible, puis le premier batch est supprimé de la liste. Réitérer ces opérations tant qu'il reste des batches.

Théorème 3.1.9 *L'heuristique H_{LBPT} fournit un ordonnancement $S_{H_{LBPT}}$ du problème $HF2(1, Bm)$ en $O(n \log n)$ avec un ratio de garantie de performance de $\frac{7}{3} - \frac{1}{3m}$, et cette borne est atteinte.*

3. rappelons que $m = \max\{m_1, m_2\}$

Preuve. De même que pour les heuristiques précédentes, la complexité de l'algorithme est donnée par les trois premières étapes qui sont des opérations de tri, d'où la complexité de l'algorithme en $O(n \log n)$. Soit $S_{H_{LBPT}}$ l'ordonnancement obtenu avec l'heuristique H_{LBPT}, nous considérons les deux étages indépendamment. Le second étage est un problème d'ordonnancement de machines parallèles à traitement par batches $(Bm|G_p = INT, b = k|C_{max})$ résolu approximativement avec un ratio de $\frac{4}{3} - \frac{1}{3m}$ par l'heuristique $\mathrm{H_{FCBLPT}}$ (Lemme 3.1.2). L'ordonnancement sans temps morts du premier étage est optimal lorsqu'il n'y a qu'une machine et que l'on ne prend pas en compte le second étage. D'où :

$$\begin{aligned} C_{max}(S_{H_{LBPT}}) &\leqslant C_{max}(S_{H_1}) + C_{max}(S_{H_{Bm}}) \\ &\leqslant C_{max}(S_1^*) + \left(\tfrac{4}{3} - \tfrac{1}{3m}\right) C_{max}(S_{Bm}^*) \end{aligned}$$

Soit S^* l'ordonnancement optimal du problème de flowshop hybride, et $C_{max}(S^*)$ sa date de fin, alors

$$C_{max}(S^*) \geqslant C_{max}(S_1^*) \qquad \text{et} \qquad C_{max}(S^*) \geqslant C_{max}(S_{Bm}^*)$$

Donc :

$$\begin{aligned} C_{max}(S_{H_{LBPT}}) &\leqslant C_{max}(S_1^*) + \left(\tfrac{4}{3} - \tfrac{1}{3m}\right) C_{max}(S_{Bm}^*) \\ &\leqslant \left(\tfrac{7}{3} - \tfrac{1}{3m}\right) C_{max}(S^*) \end{aligned}$$

C'est-à-dire :

$$\frac{C_{max}(S_{H_{LBPT}})}{C_{max}(S^*)} \leqslant \frac{7}{3} - \frac{1}{3m}$$

Ainsi lorsqu'il y a une seule machine au premier étage, l'heuristique H_{LBPT} fournit une solution avec un facteur de performance borné par $\frac{7}{3} - \frac{1}{3m}$. Nous présentons une instance pour laquelle cette borne est atteinte.

Soit I une instance du problème de flowshop hybride $HF2(1, Bm)$ pour laquelle la capacité des batches est $\max\{2; nb_C\}$. Cette instance est composée de $2m + 1$ tâches de type A, d'une tâche de type B et de nb_C tâches de type C. Les durées d'exécution des tâches sont indiquées dans le tableau 3.6, dans lequel $\epsilon < \min\{\frac{m}{3}; \frac{(m-1)^2}{2m+1}\}$. Dans l'ordonnancement optimal et dans l'ordonnancement fourni par l'heuristique la composition des batches contenant la plus petite des tâches A_1, et les tâches A_2 à A_m est identique. Dans les deux cas les batches sont unitaires car les tâches sont incompatibles 2 à 2. En revanche alors que dans l'ordonnancement fourni par l'heuristique le premier batch contient la plus grande des tâches A_1 et la tâche B dans l'ordonnancement optimal ces tâches appartiennent à des batches unitaires. Enfin les tâches de type C appartiennent à un même batch.

S'il n'y a pas de tâches de type C ($nb_C = 0$), alors la date de fin de l'ordonnancement $C_{max}(S_{H_{LBPT}})$ de l'ordonnancement $S_{H_{LBPT}}$ fournie par l'heuristique $\mathrm{H_{LBPT}}$ pour l'instance I est égal à $C_{max}(S_{H_{LBPT}}) = 7m - 1 - 2\epsilon$ (voir figure 3.11) alors que la date de fin d'un ordonnancement optimal pour l'instance I est égal à $C_{max}(S^*) = 3m + (2m + 2)\epsilon$ (voir figure 3.12). De plus $C_{max}(S^*)$ est optimal car il atteint la borne inférieure $LB_2 = \frac{\sum_{j \in J} p_j}{m} + \min_{j \in J} a_j$. D'où :

$$\frac{C_{max}(S_{H_{LBPT}})}{C_{max}(S^*)} = \frac{7m - 1 - 2\epsilon}{3m + (2m + 2)\epsilon}$$

Ainsi :

$$\lim_{\epsilon \to 0} \frac{C_{max}(S_{H_{LBPT}})}{C_{max}(S^*)} = \frac{7}{3} - \frac{1}{3m}$$

Si $nb_C > 0$, lorsque ϵ tend vers 0 les tâches de type C deviennent insignifiantes.

□

Type	Nombre de tâches	p_1	p_2
A_1	2	ϵ	$[2m-1;2m-1]$ et $[2m-1-\epsilon;2m-1-\epsilon]$
A_2	2	ϵ	$[2m-2;2m-2]$ et $[2m-2-\epsilon;2m-2-\epsilon]$
		...	
A_{m-1}	2	ϵ	$[m+1;m+1]$ et $[m+1-\epsilon;m+1-\epsilon]$
A_m	3	ϵ	$[m;m]$, $[m-\epsilon;m-\epsilon]$ et $[m-2\epsilon;m-2\epsilon]$
B	1	$3m$	$[\epsilon,2m]$
C	nb_C	ϵ	$[\epsilon;\epsilon]$

TABLE 3.6 – Durées d'exécution des tâches de l'instance I (H_{LPBT} pour $HF2(1,Bm)$).

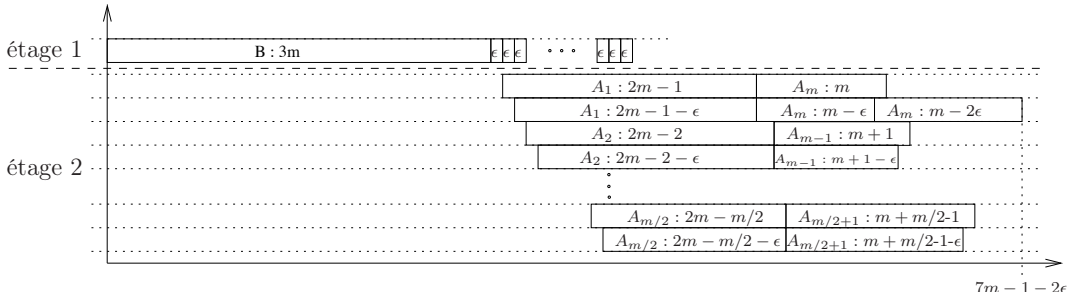


FIGURE 3.11 – Ordonnancement $S_{H_{LBPT}}$ (pour $HF2(1,Bm)$).

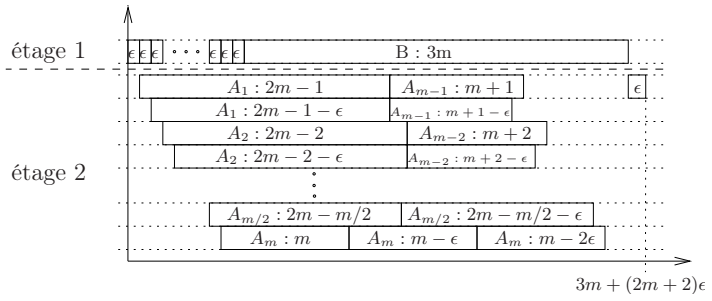


FIGURE 3.12 – Ordonnancement optimal S^* (H_{LBPT} pour $HF2(1,Bm)$).

L'heuristique inverse $\mathrm{H}^{\mathrm{Inv}}_{\mathrm{LBPT}}$ respecte le schéma général des heuristiques inverses, présenté dans la section 3.1.1. Elle utilise la règle LPT pour trier la liste de batches.

Théorème 3.1.10 *L'heuristique H^{Inv}_{LBPT} fournit un ordonnancement $S^{Inv}_{H_{LBPT}}$ du problème $HF2$ $(Bm,1)$ en $O(n\log n)$ avec un ratio de garantie de performance de $\frac{7}{3}-\frac{1}{3m}$, et cette borne est atteinte.*

Preuve. La preuve est identique à celle de l'heuristique $\mathrm{H}_{\mathrm{LBPT}}$. □

Heuristique H_J (Johnson). Au second étage, les batches sont créés suivant la règle FCBLPT et triés à l'aide de la règle de Johnson [111], que nous adaptons à notre problème. Les tâches du premier batch sont exécutées sur la machine du premier étage sans temps morts. Dès que toutes les tâches du premier batch sont disponibles au second étage, le batch est exécuté sur la première machine disponible, puis le premier batch est supprimé de la liste. Réitérer ces opérations tant qu'il reste des batches.

Théorème 3.1.11 *L'heuristique H_J fournit un ordonnancement S_{H_J} du problème $HF2(1, Bm)$ en $O(n \log n)$ avec un ratio de garantie de performance de $3 - \frac{1}{m}$.*

Preuve. De même que pour les heuristiques précédentes, la complexité de l'algorithme est donnée par les trois premières étapes qui sont des opérations de tri, d'où la complexité de l'algorithme en $O(n \log n)$. Soit S_{H_J} l'ordonnancement obtenu avec l'heuristique H_J, nous considérons les deux étages indépendamment. L'ordonnancement du second étage ne respecte plus la règle LPT, mais c'est un ordonnancement de liste, or le problème d'ordonnancement de machines parallèles à traitement par batches $(Bm|G_p = INT, b = k|C_{max})$ est résolu approximativement avec un ratio de $2 - \frac{1}{m}$ par l'heuristique $\mathrm{H_{FCBList}}$ (Lemme 3.1.1). L'ordonnancement sans temps mort du premier étage est optimal lorsqu'il n'y a qu'une machine et que l'on ne prend pas en compte le second étage. D'où :

$$\begin{aligned} C_{max}(S_{H_J}) &\leqslant C_{max}(S_{H_1}) + C_{max}(S_{H_{Bm}}) \\ &\leqslant C_{max}(S_1^*) + \left(2 - \tfrac{1}{m}\right) C_{max}(S_{Bm}^*) \end{aligned}$$

Soit S^* l'ordonnancement optimal du problème de flowshop hybride, et $C_{max}(S^*)$ sa date de fin, alors

$$C_{max}(S^*) \geqslant C_{max}(S_1^*) \qquad \text{et} \qquad C_{max}(S^*) \geqslant C_{max}(S_{Bm}^*)$$

Donc :

$$\begin{aligned} C_{max}(S_{H_J}) &\leqslant C_{max}(S_1^*) + \left(2 - \tfrac{1}{m}\right) C_{max}(S_{Bm}^*) \\ &\leqslant \left(3 - \tfrac{1}{m}\right) C_{max}(S^*) \end{aligned}$$

C'est-à-dire :

$$\frac{C_{max}(S_{H_J})}{C_{max}(S^*)} \leqslant 3 - \frac{1}{m}$$

L'existence d'une instance pour laquelle cette borne est atteinte est une question ouverte.
□

L'heuristique inverse $\mathrm{H_J^{Inv}}$ respecte le schéma général des heuristiques inverses, présenté dans la section 3.1.1, en utilisant la règle de Johnson pour trier la liste de batches.

Théorème 3.1.12 *L'heuristique H_J^{Inv} fournit un ordonnancement $S_{H_J^{Inv}}$ du problème $HF2(Bm, 1)$ en $O(n \log n)$ avec un ratio de garantie de performance de $3 - \frac{1}{m}$.*

Preuve. La preuve est identique à celle de l'heuristique $\mathrm{H_J}$. □

C. Une machine au second étage

Finalement, nous traitons le cas où une seule machine est présente sur le second étage, ce problème est noté $HF2(m, B)$. À l'instar du problème avec plusieurs machines sur chaque étage, nous regroupons les heuristiques par paire, la première heuristique de chaque paire est développée pour le problème initial $HF2(m, B)$ alors que la seconde est développée pour le problème inverse $HF2(B, m)$. Le premier type d'heuristique, est une adaptation des heuristiques $\mathrm{H_{LPT}}$ du problème général, ainsi les tâches sont ordonnancées suivant les règles LPT et FAM, puis les batches sont exécutés dès que toutes leurs tâches sont disponibles pour la machine du second étage. Le second type d'heuristique, adapté des heuristiques $\mathrm{H_J}$ du problème général trie les batches suivant la règle de Johnson puis ordonnance les tâches batch après batch sur la première machine disponible.

Heuristique H_{LPT}. Au premier étage les tâches sont ordonnancées suivant les règles LPT et FAM. Au second étage les batches sont créés suivant la règle FCBLPT, puis triés par dates de disponibilité au second étage croissantes. Dès que toutes les tâches du premier batch sont disponibles au second étage, le batch est exécuté sur la machine, et supprimé de la liste. Réitérer cette opération tant qu'il reste des batches dans la liste.

Théorème 3.1.13 *L'heuristique H_{LPT} fournit un ordonnancement $S_{H_{LPT}}$ du problème $HF2(m, B)$ en $O(n \log n)$ avec un ratio de garantie de performance de $\frac{7}{3} - \frac{1}{3m}$, et cette borne est atteinte.*

Preuve. La complexité de l'algorithme est donnée, comme pour les autres heuristiques, par les trois premières étapes qui sont des opérations de tri, d'où la complexité de l'algorithme en $O(n \log n)$. Soit $S_{H_{LPT}}$ l'ordonnancement obtenu avec l'heuristique H_{LPT}, nous considérons les deux étages indépendamment. Au premier étage la règle LPT fournit un ordonnancement avec un ratio de performance garantie égal à $\frac{4}{3} - \frac{1}{3m}$. Au second étage, le problème d'ordonnancement sur une machine à traitement par batches $(B|G_p = INT, b = k|C_{max})$ est résolu optimalement grâce à la règle FCBLPT (voir Oulamara et al. [176]). D'où :

$$\begin{aligned} C_{max}(S_{H_{LPT}}) &\leqslant C_{max}(S_{H_{Pm}}) + C_{max}(S_{H_B}) \\ &\leqslant \left(\tfrac{4}{3} - \tfrac{1}{3m}\right) C_{max}(S^*_{Pm}) + C_{max}(S^*_B) \end{aligned}$$

Soit S^* l'ordonnancement optimal du problème de flowshop hybride, et $C_{max}(S^*)$ sa date de fin, alors

$$C_{max}(S^*) \geqslant C_{max}(S^*_{Pm}) \qquad \text{et} \qquad C_{max}(S^*) \geqslant C_{max}(S^*_B)$$

Donc :

$$\begin{aligned} C_{max}(S_{H_{LPT}}) &\leqslant \left(\tfrac{4}{3} - \tfrac{1}{3m}\right) C_{max}(S^*_{Pm}) + C_{max}(S^*_B) \\ &\leqslant \left(\tfrac{7}{3} - \tfrac{1}{3m}\right) C_{max}(S^*) \end{aligned}$$

C'est-à-dire :

$$\frac{C_{max}(S_{H_{LPT}})}{C_{max}(S^*)} \leqslant \frac{7}{3} - \frac{1}{3m}$$

Ainsi lorsqu'il y a une seule machine sur le second étage, l'heuristique H_{LPT} fournit une solution avec un facteur de performance borné par $\frac{7}{3} - \frac{1}{3m}$. Nous présentons une instance pour laquelle cette borne est atteinte.

Soit I une instance du problème de flowshop hybride $HF2(m, B)$, pour laquelle la capacité des batches est égale à $2m + 2$. Cette instance est composée de $2m + 1$ tâches de type A, d'une

tâche de type B et de nb_C tâches de type C. Les durées d'exécution des tâches sont indiquées dans le tableau 3.7, dans lequel $\epsilon < \min\{\frac{m}{3}; \frac{(m-1)^2}{2m+1}\}$.

Type	Nombre de tâches	p_1	p_2
A_1	2	$2m-1$	$[\epsilon; 3m]$
A_2	2	$2m-2$	$[\epsilon; 3m]$
	...		
A_{m-1}	2	$m+1$	$[\epsilon; 3m]$
A_m	3	m	$[\epsilon; 3m]$
B	1	ϵ	$[3m; 3m]$
C	nb_C	ϵ	$[\epsilon; \epsilon]$

TABLE 3.7 – Durées d'exécution des tâches de l'instance I (H_{LPT} pour $HF2(m, B)$).

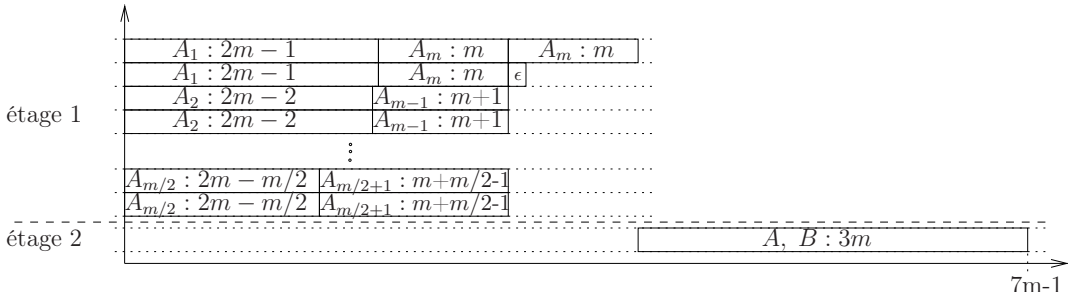


FIGURE 3.13 – Ordonnancement $S_{H_{LPT}}$ (pour $HF2(m, B)$).

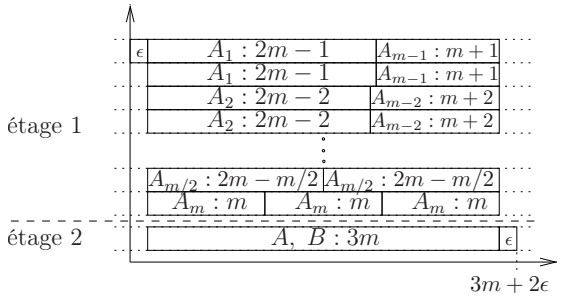


FIGURE 3.14 – Ordonnancement optimal S^* (H_{LPT} pour $HF2(m, B)$).

S'il n'y a pas de tâches de type C ($nb_C = 0$), alors la date de fin de l'ordonnancement $C_{max}(S_{H_{LPT}})$ de l'ordonnancement $S_{H_{LPT}}$ fournie par l'heuristique $\mathrm{H_{LPT}}$ pour l'instance I est égal à $C_{max}(S_{H_{LPT}}) = 7m - 1$ (voir figure 3.13) alors que la date de fin d'un ordonnancement optimal pour l'instance I est égal à $C_{max}(S^*) = 3m + 2\epsilon$ (voir figure 3.14). De plus $C_{max}(S^*)$ est optimal car il atteint la borne inférieure $LB_2 = \frac{\sum_{j \in J} p_j}{m} + \min_{j \in J} a_j$. D'où :

$$\frac{C_{max}(S_{H_{LPT}})}{C_{max}(S^*)} = \frac{7m-1}{3m+2\epsilon}$$

Ainsi :

$$\lim_{\epsilon \to 0} \frac{C_{max}(S_{H_{LPT}})}{C_{max}(S^*)} = \frac{7}{3} - \frac{1}{3m}$$

Si $nb_C > 0$, lorsque ϵ tend vers 0 les tâches de type C deviennent insignifiantes.

□

Dans ce cas précis une méthode inverse est inutile, en effet l'ordre d'exécution des batches dépendant de l'exécution des tâches qui seraient séquencées plus tard.

Heuristique H_J (Johnson). Au second étage, les batches sont créés suivant la règle FCBLPT et triés à l'aide de la règle de Johnson [111], que nous adaptons à notre problème. Les tâches du premier batch sont exécutées au premier étage suivant les règle LPT et FAM. Dès que toutes les tâches du premier batch sont disponibles au second étage, le batch est exécuté sur la machine, puis le premier batch est supprimé de la liste. Réitérer ces opérations tant qu'il reste des batches.

Théorème 3.1.14 *L'heuristique H_J fournit un ordonnancement S_{H_J} du problème $HF2(m, Bm)$ en $O(n \log n)$ avec un ratio de garantie de performance de $3 - \frac{1}{m}$.*

Preuve. De même que pour les heuristiques précédentes, la complexité de l'algorithme est donnée par les trois premières étapes qui sont des opérations de tri, d'où la complexité de l'algorithme en $O(n \log n)$. Soit S_{H_J} l'ordonnancement obtenu avec l'heuristique H_J, nous considérons les deux étages indépendamment. Au premier étage, l'ordonnancement de liste fournit des ordonnancements pour le problème de machines classiques parallèles $(Pm||C_{max})$ avec un ratio de garantie de performance de $2 - \frac{1}{m}$. Au second étage, le problème d'ordonnancement sur une machine à traitement par batches $(B|G_p = INT, b = k|C_{max})$ est résolu optimalement grâce à la règle FCBLPT (voir Oulamara et al. [176]). D'où :

$$\begin{aligned} C_{max}(S_{H_J}) &\leqslant C_{max}(S_{H_{Pm}}) + C_{max}(S_{H_B}) \\ &\leqslant \left(2 - \tfrac{1}{m}\right) C_{max}(S^*_{Pm}) + C_{max}(S^*_B) \end{aligned}$$

Soit S^* l'ordonnancement optimal du problème de flowshop hybride, et $C_{max}(S^*)$ sa date de fin, alors

$$C_{max}(S^*) \geqslant C_{max}(S^*_{Pm}) \qquad \text{et} \qquad C_{max}(S^*) \geqslant C_{max}(S^*_B)$$

Donc :

$$\begin{aligned} C_{max}(S_{H_J}) &\leqslant \left(2 - \tfrac{1}{m_1}\right) C_{max}(S^*_{Pm}) + C_{max}(S^*_B) \\ &\leqslant \left(3 - \tfrac{1}{m}\right) C_{max}(S^*) \end{aligned}$$

C'est-à-dire :

$$\frac{C_{max}(S_{H_J})}{C_{max}(S^*)} \leqslant 3 - \frac{1}{m}$$

Ainsi lorsqu'il y a une seule machine sur le second étage, l'heuristique H_J fournit une solution avec un facteur de performance borné par $3 - \frac{1}{m}$. Nous présentons une instance pour laquelle cette borne est atteinte.

Soit I une instance du problème de flowshop hybride $HF2(m, B)$, pour laquelle la capacité des batches est égale à $2m$. Cette instance est composée de $2m+1$ tâches de type A, d'une tâche de type B et de nb_C tâches de type C. Les durées d'exécution des tâches sont indiquées dans le tableau 3.8, dans lequel $\epsilon < \min\{\frac{m}{3}; \frac{(m-1)^2}{2m+1}\}$.

Type	Nombre de tâches	p_1	p_2
A	$m-1$	$(m-1)L$	$[\epsilon; mL]$
B	$m-1$	L	$[\epsilon; mL]$
C	1	mL	$[\epsilon; mL]$
D	1	ϵ	$[mL; mL]$
E	nb_E	ϵ	$[\epsilon/2; \epsilon/2]$

TABLE 3.8 – Durées d'exécution des tâches de l'instance I (H_J pour $HF2(m, B)$).

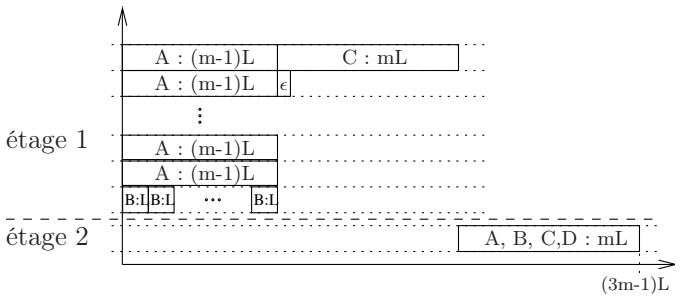


FIGURE 3.15 – Ordonnancement S_{H_J} (pour $HF2(m, B)$).

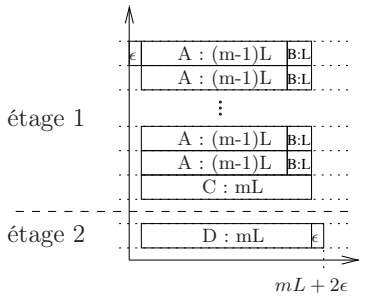


FIGURE 3.16 – Ordonnancement optimal S^* (H_J pour $HF2(m, B)$).

S'il n'y a pas de tâches de type E ($nb_E = 0$), alors la date de fin de l'ordonnancement $C_{max}(S_{H_J})$ de l'ordonnancement S_{H_J} fournie par l'heuristique H_J pour l'instance I est égal à $C_{max}(S_{H_J}) = (3m-1)L$ (voir figure 3.15) alors que la date de fin d'un ordonnancement optimal pour l'instance I est égal à $C_{max}(S^*) = mL+2\epsilon$ (voir figure 3.16). De plus $C_{max}(S^*)$ est optimal car il atteint la borne inférieure $LB_2 = \frac{\sum_{j\in J} p_j}{m} + \min_{j\in J} a_j$. D'où :

$$\frac{C_{max}(S_{H_J})}{C_{max}(S^*)} = \frac{(3m-1)L}{mL+2\epsilon}$$

Ainsi :

$$\lim_{\epsilon \to 0} \frac{C_{max}(S_{H_J})}{C_{max}(S^*)} = 3 - \frac{1}{m}$$

Si $nb_E > 0$, lorsque ϵ tend vers 0 les tâches de type E deviennent insignifiantes.

□

L'heuristique inverse $\mathrm{H}_{\mathrm{J}}^{\mathrm{Inv}}$ respecte le schéma général des heuristiques inverses, présenté dans la section 3.1.1, en utilisant la règle de Johnson pour trier la liste de batches.

Théorème 3.1.15 *L'heuristique H_J^{Inv} fournit un ordonnancement $S_{H_J^{Inv}}$ du problème $HF2(B, m)$ en $O(n \log n)$ avec un ratio de garantie de performance de $3 - \frac{1}{m}$.*

Preuve. La preuve est identique à celle de l'heuristique H_{J}. □

3.1.5 Résultats expérimentaux

Dans cette section, nous présentons les performances de ces heuristiques sur plusieurs types d'instances de 200 tâches. Nous désignons par H^* la méthode qui consiste à appliquer les 6 heuristiques et à retenir la meilleure solution trouvée. L'indicateur d'efficacité, d'une heuristique H, utilisé ici est l'écart relatif entre la valeur de la solution obtenue C_{max}^H et la borne inférieure LB :

$$\frac{C_{max}^H - LB}{LB} \times 100$$

Pour chaque type d'instances nous présentons ainsi l'écart relatif moyen, et l'écart relatif maximum pour chaque heuristique (H^* incluse). Nous accompagnons ces valeurs du nombre d'instances (sur 10 pour les tableaux A.1, A.2, A.3, A.4) pour lesquelles l'heuristique a été la meilleure des 6 heuristiques (colonne *"Best"*).

Nous avons mené les expériences sur des instances de 200 tâches, pour lesquelles les durées d'exécution sur le premier étage sont tirées aléatoirement dans l'intervalle $[15, 185]$ selon une distribution uniforme. Au second étage les intervalles de durées d'exécution sont proportionnels de la forme $[a_i, (1+\alpha)a_i]$, la valeur de α est égale à 0,1 ou 1 et les durées minimales d'exécution, a_j sont tirées aléatoirement dans l'un des intervalles $[15, 185]$, $[30, 370]$, $[45, 555]$ ou $[60, 740]$. C'est-à-dire dans les intervalles de la forme $[15c_2, 185c_2]$ avec $c_2 \in \{1, 2, 3, 4\}$, où c_2 indique le rapport moyen entre les durées d'exécution des deux étages. La capacité des batches est égale à 2, 3 ou 10. Les combinaisons de nombre de machines sont notées $m_1 - m_2$, telles que m_1 représente le nombre de machines sur le premier étage, et m_2 le nombre de machines sur le second étage de type batch. Les combinaisons utilisées sont : 1-1, 1-10, 10-1, 10-10 et 2-2, 2-5, 5-2, 5-5. Chaque combinaison de caractéristiques est testée sur 10 instances, et nous présentons un tableau pour chaque valeur de c_2.

À la lecture des tableaux A.1, A.2, A.3 et A.4 (voir Annexe A), nous remarquons tout d'abord qu'il y a peu de différences de qualité entre les différentes valeurs de α. C'est pourquoi nous étudions les résultats du tableau 3.9, dans lequel nous avons rassemblé les différentes valeurs de $\alpha = \{0,1;\ 1\}$ en calculant, respectivement, les moyennes, maximums et sommes pour $\alpha = \{0,1;\ 1\}$ des colonnes "Moy", "Max" et "Best". Nous avons également décidé d'étudier, en détail, seulement les instances avec 2 ou 5 machines sur chaque étage. En effet, à la lecture des tableaux A.1, A.2, A.3 et A.4 nous remarquons que lorsqu'il y a une seule machine sur le premier étage l'écart relatif est toujours inférieur à 0,75%, avec une moyenne inférieure à 0,1%. De plus, dans le cas où le premier étage est composé de 10 machines et le second d'une seule machine, lorsque le premier étage est plus critique la meilleure solution s'éloigne de la borne inférieure. Cependant, hormis lorsque la capacité des batches est égale à 10, les résultats sont relativement proches de la borne inférieure. Quant-aux instances avec 10 machines sur chaque étage, elles confirment les résultats que nous présentons pour les cas avec 2 et 5 machines sur chaque étage.

Les résultats du tableau 3.9, exhibent l'importance du nombre de machines du premier étage sur la qualité de la meilleure solution obtenue. En effet, tous les écarts relatifs de l'heuristique H^* pour les combinaisons avec 2 machines sur le premier étage sont inférieurs à leur équivalent à 5 machines sur le premier étage. En revanche, aucune tendance comparable ne se dégage lorsque le nombre de machines sur le second étage varie. Finalement remarquons qu'aucune tendance ne se dégage quant-à une relation entre la valeur de l'écart relatif (par rapport à la borne inférieure) et la capacité des batches ou la valeur de c_2. Les résultats fournis par l'heuristique H^* sont donc relativement homogènes.

Malgré cette homogénéité de la qualité de la meilleure solution, la qualité de chaque heuristique est variable. Ainsi chaque heuristique est plus ou moins intéressante par rapport aux autres heuristiques, hormis H_{LPT} qui est toujours dominée par les autres heuristiques (nous avons ajouté un fond gris aux colonnes correspondantes). C'est pourquoi nous comparons les instances pour lesquelles chaque heuristique a fourni la meilleure valeur (égale à H^*), comparaison que nous effectuons par combinaison de machines en fonction de c_2 et de k. Cependant, le saut de valeur entre $k = 3$ et $k = 10$ étant trop important pour pouvoir analyser les causes des variations, nous restreignons la comparaison à $k = 2$ et $k = 3$. Notons toutefois que lorsque $k = 10$, hormis pour 3 types d'instances pour lesquels la tendance est moins marquée, l'heuristique H^* fournit, pour chaque type d'instances, des solutions avec une moyenne significativement meilleure que celles de chaque heuristique utilisée séparément.

Ainsi, la lecture du tableau 3.9 nous permet de remarquer que le comportement relatif des heuristiques, lorsque le nombre de machines est le même sur chaque étage, est stable, en effet dans les 3 cas (2-2, 5-5 et 10-10) l'heuristique H_J^{Inv} fournit la plupart du temps les meilleures solutions lorsque $k = 2$ et $c_2 = 1$ et lorsque $k = 3$ et $c_2 = \{1, 2\}$. Dans les autres cas, c'est l'heuristique H_J qui fournit les meilleurs résultats. Dans ce cas, les heuristiques basées sur $LBPT$ fournissent très peu de meilleures solutions qui ne sont pas obtenues également par les heuristiques basées sur la règle de Johnson. En revanche lorsqu'il y a 2 machines sur le premier étage et 5 sur le second et lorsque $c_2 = 1$, les meilleures solutions sont partagées entre les 4 heuristiques ; cependant pour les autres valeurs de c_2, lorsque $k = 2$ l'heuristique H_J^{Inv} fournit la majorité des meilleures solutions et pour $k = 3$ les meilleures solutions sont partagées principalement entre les heuristiques H_J^{Inv} et H_{LBPT}. Finalement, lorsqu'il y a 5 machines sur le premier étage et 2 sur le second nous remarquons que, sauf pour le cas où $k = 3$ et $c_2 = 1$, l'heuristique H_J fournit les meilleures solutions. Exceptionnellement les meilleures solutions peuvent être obtenues par l'heuristique H_J^{Inv}.

Aux vues de ces résultats nous pensons qu'il existe, pour chaque combinaison de nombre de machines, une relation entre le ratio k/c_2 et la meilleure heuristique à utiliser. De plus cette relation semble logique car le ratio k/c_2 est un indicateur du rapport entre les charges des deux étages pour une combinaison identique du nombre de machines. Aux vues des résultats concernant les instances avec le même nombre de machines sur chaque étage, il semblerait que l'on puisse étendre ces résultats à ratio $\frac{m_2}{m_1}$ constant.

Finalement, le tableau 3.10 nous permet de confirmer l'avantage des méthodes utilisant la règle de Johnson. Contrairement à la règle LPT, la règle de Johnson prend en compte les spécificités des flowshops ce qui explique cet avantage. Ainsi, l'heuristique H_J fournit la meilleure solution dans 70% des cas. Finalement, la combinaison de ces heuristiques, aux temps de calcul instantanés, permet, pour des instances de 200 tâches, d'obtenir en moyenne une solution à moins de 1% de la borne inférieure. Dans le pire cas traité ici, l'écart par rapport à la borne inférieure est de 8%.

m_1-m_2	k	c_2	H_{LPT}						H_J						H_{LBPT}						H^*	
			Normal			Inverse			Normal			Inverse			Normal			Inverse				
			Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max
2–2	2	1	51,01	59,43	0	2,46	3,71	0	0,36	0,81	3/20	***0,10***	0,23	***14***/20	0,36	0,78	2/20	0,34	0,70	3/20	0,08	0,16
		2	96,58	98,95	0	21,10	25,35	0	***0,37***	0,64	***12***/20	0,56	1,48	6/20	1,01	2,32	0	1,16	2,39	2/20	0,26	0,54
		3	66,04	72,37	0	9,60	13,86	0	***0,29***	0,51	***18***/20	0,82	1,59	2/20	1,12	1,89	0	1,62	2,06	0	0,29	0,50
		4	49,76	55,53	0	4,67	7,18	0	***0,25***	0,40	***15***/20	0,55	1,20	6/20	0,85	1,23	0	1,48	1,81	0	0,23	0,38
	3	1	32,93	34,53	0	2,10	2,64	0	0,42	0,82	5/20	***0,27***	0,67	***13***/20	0,41	0,81	3/20	0,43	0,87	6/20	0,25	0,58
		2	67,25	70,67	0	6,56	9,81	0	0,33	0,78	3/20	0,20	0,35	***11***/20	0,31	0,82	3/20	0,30	0,71	6/20	0,13	0,30
		3	95,77	99,14	0	22,89	26,77	0	***0,74***	1,33	***15***/20	1,07	2,65	4/20	1,71	3,04	0	2,05	3,57	1/20	0,66	1,33
		4	73,91	77,72	0	12,96	16,24	0	***0,80***	1,48	***14***/20	1,23	2,82	5/20	1,89	2,93	1/20	2,70	3,66	0	0,75	1,29
	10	1	8,43	10,10	0	1,99	2,31	0	0,35	0,65	9/20	0,39	0,82	11/20	0,35	0,65	9/20	0,39	0,82	11/20	**0,25**	0,56
		2	20,11	24,67	0	3,79	4,35	0	0,58	1,05	8/20	0,40	0,87	10/20	0,45	1,05	9/20	0,41	0,87	10/20	**0,30**	0,65
		3	30,73	35,51	0	5,33	6,05	0	0,42	0,99	11/20	0,56	1,13	7/20	0,42	0,99	12/20	0,56	1,13	6/20	**0,32**	0,67
		4	41,33	45,62	0	6,95	7,79	0	0,53	1,36	7/20	0,54	1,03	6/20	0,49	1,11	11/20	0,50	1,03	8/20	**0,37**	0,91
2–5	2	1	18,16	20,38	0	1,97	2,30	0	0,29	0,63	**8**/20	0,28	0,75	**9**/20	0,30	0,63	**7**/20	0,36	0,75	**7**/20	0,15	0,28
		2	37,06	40,34	0	3,66	4,14	0	0,38	0,74	1/20	***0,14***	0,36	***14***/20	0,38	0,73	2/20	0,45	0,88	4/20	0,11	0,36
		3	59,42	65,99	0	6,70	8,63	0	0,43	0,92	3/20	***0,13***	0,25	***15***/20	0,43	0,92	1/20	0,35	0,82	3/20	0,11	0,25
		4	79,19	86,07	0	15,29	21,93	0	0,29	0,72	4/20	***0,11***	0,21	***16***/20	0,28	0,75	3/20	0,45	0,81	0	0,10	0,21
	3	1	10,69	11,87	0	2,12	2,63	0	0,46	0,87	**6**/20	0,35	0,75	**12**/20	0,45	0,87	**8**/20	0,35	0,75	**12**/20	0,23	0,46
		2	24,06	26,78	0	3,71	4,20	0	0,36	0,78	**8**/20	0,33	0,90	**10**/20	0,34	0,78	**9**/20	0,40	0,90	**6**/20	0,20	0,51
		3	37,07	41,36	0	5,33	5,88	0	0,32	0,71	3/20	0,21	0,55	**9**/20	0,31	0,73	**7**/20	0,34	0,75	4/20	0,10	0,21
		4	51,28	57,12	0	7,21	8,75	0	0,42	0,77	3/20	0,22	0,53	**11**/20	0,41	0,75	**6**/20	0,37	0,86	5/20	0,18	0,37
	10	1	1,91	2,42	0	2,09	2,58	0	0,44	0,97	10/20	0,40	0,84	11/20	0,44	0,97	10/20	0,40	0,84	11/20	**0,28**	0,65
		2	3,72	4,13	0	3,72	4,16	0	0,44	0,94	12/20	0,49	0,72	9/20	0,44	0,94	12/20	0,49	0,72	9/20	**0,30**	0,66
		3	5,35	6,25	0	5,30	6,05	0	0,38	0,81	15/20	0,57	0,89	5/20	0,38	0,81	15/20	0,57	0,89	5/20	0,35	0,73
		4	7,87	10,04	0	6,96	7,58	0	0,47	1,06	13/20	0,58	1,00	6/20	0,47	1,06	13/20	0,58	1,00	7/20	0,41	0,83
5–2	2	1	78,23	88,90	0	14,79	20,90	0	***0,38***	0,69	***19***/20	0,73	1,50	1/20	2,06	3,47	0	2,77	4,40	0	0,37	0,69
		2	40,05	43,54	0	3,33	4,70	0	***0,27***	0,50	***19***/20	0,90	1,73	1/20	1,04	1,64	0	1,72	2,45	0	0,27	0,50
		3	26,66	29,30	0	1,41	2,28	0	***0,28***	0,92	***17***/20	0,77	1,52	3/20	0,77	1,12	0	1,62	2,06	0	0,24	0,46
		4	19,92	21,90	0	0,85	1,15	0	***0,36***	1,09	***14***/20	0,63	1,42	5/20	0,62	0,87	1/20	1,47	1,81	0	0,26	0,73
	3	1	83,99	88,53	0	15,98	21,18	0	1,84	2,79	0	***0,82***	1,29	***19***/20	1,84	2,63	1/20	1,87	2,82	0	0,81	1,29
		2	59,02	63,60	0	8,38	11,79	0	***0,88***	1,78	***19***/20	1,65	3,15	1/20	1,97	2,79	0	3,09	4,04	0	0,88	1,48
		3	39,71	42,87	0	3,76	4,67	0	***0,57***	0,87	***19***/20	1,29	2,67	1/20	1,50	1,89	0	2,53	3,05	0	0,57	0,87
		4	29,90	31,45	0	1,88	2,76	0	0,69	1,87	***14***/20	1,06	2,56	5/20	1,16	1,72	1/20	2,38	2,73	0	0,57	1,06
	10	1	25,63	29,31	0	5,72	6,57	0	1,63	2,39	11/20	1,73	2,81	8/20	1,65	2,39	12/20	1,73	2,81	9/20	**1,42**	2,35
		2	56,00	65,91	0	10,25	11,32	0	1,87	3,09	6/20	1,67	2,58	11/20	1,88	3,09	6/20	2,03	3,10	5/20	**1,46**	2,20
		3	80,50	94,61	0	17,98	23,96	0	2,08	3,32	4/20	1,81	3,40	8/20	2,20	4,24	3/20	2,19	4,14	6/20	**1,35**	2,44
		4	88,52	97,35	0	21,74	32,06	0	4,66	8,51	12/20	5,23	9,93	6/20	7,54	8,95	3/20	9,08	10,84	0	4,42	7,68
5–5	2	1	49,40	56,52	0	6,07	8,18	0	1,48	2,42	0	***0,53***	1,33	***17***/20	1,48	2,41	2/20	1,46	2,46	1/20	0,51	1,70
		2	94,22	97,82	0	25,82	32,71	0	1,54	2,71	***14***/20	2,86	6,00	5/20	3,42	5,01	1/20	5,22	7,55	0	1,19	1,96
		3	64,38	71,74	0	12,18	17,94	0	***0,93***	1,51	***20***/20	3,59	5,80	0	2,59	3,54	0	5,01	6,25	0	0,93	1,51
		4	48,77	54,54	0	6,68	8,00	0	***0,82***	1,18	***20***/20	3,07	5,42	0	2,26	2,94	0	4,93	5,72	0	0,82	1,18
	3	1	31,56	34,30	0	5,90	7,68	0	1,74	3,10	7/20	1,48	2,98	***11***/20	1,74	3,13	6/20	1,61	2,97	3/20	1,24	1,99
		2	64,86	70,50	0	11,98	14,50	0	1,73	2,36	3/20	***1,01***	1,78	***16***/20	1,70	2,34	4/20	1,77	3,01	1/20	0,95	1,67
		3	94,40	97,50	0	27,66	32,52	0	***2,34***	3,38	***17***/20	4,16	7,75	3/20	4,63	6,92	0	7,15	10,01	0	2,18	3,38
		4	72,38	80,00	0	16,46	20,35	0	***2,39***	3,08	***20***/20	5,63	8,17	0	4,74	5,44	0	8,08	9,71	0	2,39	3,08
	10	1	5,64	6,94	0	6,03	6,85	0	1,92	2,94	5/20	1,62	2,52	15/20	1,93	2,94	5/20	1,63	2,52	14/20	**1,49**	2,50
		2	13,48	15,54	0	9,74	11,16	0	1,59	2,88	13/20	1,85	3,18	6/20	1,58	2,88	14/20	1,83	3,18	6/20	**1,36**	2,69
		3	23,63	28,56	0	13,87	16,98	0	1,73	3,34	12/20	2,14	3,17	7/20	1,72	3,34	12/20	2,13	3,17	5/20	**1,53**	2,80
		4	34,10	38,96	0	17,75	19,23	0	1,83	3,24	10/20	2,01	3,65	8/20	1,82	3,03	10/20	2,08	3,16	5/20	**1,53**	2,95

TABLE 3.9 – Comparaison des heuristiques pour $HF2(m_1, Bm_2)$.

H_{LPT}						H_J						H_{LBPT}						H^*	
Normal			Inverse			Normal			Inverse			Normal			Inverse				
Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max
39,50	99,14	0	8,71	43,37	1%	1,16	8,51	70%	1,55	19,24	60%	1,64	14,06	37%	2,21	19,66	33%	0,97	8,02

TABLE 3.10 – Comparaison des heuristiques pour $HF2(m_1, Bm_2)$ (moyenne globale).

3.2 Durées d'exécution identiques au premier étage

Dans cette section nous revenons à nouveau au problème général, à savoir plusieurs machines sur chaque étage, et nous étudions le cas particulier où les tâches ont des durées d'exécution identiques sur le premier étage. Nous avons également supposé que les intervalles de durées d'exécution du second étage sont uniformes, c'est-à-dire que pour toutes tâches i et j telles que $a_i \leqslant a_j$ alors $b_i \leqslant b_j$. L'étude de ce cas est justifiée par le problème industriel (voir 1.4) où les temps de constructions sont semblables et où un pneu à un temps de cuisson donné avec une divergence acceptée de quelques pourcents (4% dans notre cas). C'est-à-dire des intervalles de durées d'exécution proportionnels auxquels nous avons préférés les intervalles uniformes qui permettent de traiter un problème plus général. Nous proposons ici un schéma d'approximation polynomial (*PTAS : Polynomial-Time-Approximation-Scheme* en anglais) pour résoudre ce problème, noté $HF2(m_1, Bm_2)|G_p^U = INT, b = k, p_j = p|C_{max}$.

Li et al. [140] ont développé un tel schéma d'approximation polynomial (PTAS) pour le problème de minimisation du makespan sur des machines parallèles à traitement par batches et dates de disponibilité, noté $Bm|b = k, r_j|C_{max}$. Notons également que Afrati et al. [2], Hall et Shmoys [97] et Deng et al. [57] ont introduit le principe et certains résultats intéressants pour ce type de schémas d'approximation polynomiaux.

En l'absence de dates de disponibilité, il existe un ordonnancement du premier étage sans temps mort, de plus les durées d'exécution des tâches au premier étage sont identiques donc à la date p chaque machine du premier étage a exécuté une tâche, puis à la date $2p$ chacune a exécuté une deuxième tâche. Au second étage nous pouvons donc considérer qu'à la date p, m_1 tâches sont disponibles, puis à la date $2p$ il y a m_1 nouvelles tâches disponibles. C'est pourquoi nous considérons les opérations du premier étage comme des dates de disponibilité généralisées, nous pouvons transformer ce problème $(HF2(m_1, Bm_2)|G_p^U = INT, b = k, p_j = p|C_{max})$ en un problème d'ordonnancement de machines parallèles à traitement par batches et dates de disponibilité généralisées. C'est-à-dire qu'à la date de disponibilité généralisée $\bar{r}_i$, $nb_{\bar{r}_i}$ tâches sont disponibles. Dans notre cas, à la première date de disponibilité généralisée $\bar{r}_1$ on aura m_1 tâches disponibles, en effet m_1 tâches sont terminées sur le premier étage et disponibles pour le second étage. À la seconde date de disponibilité $\bar{r}_2$, au total, $2m_1$ tâches sont disponibles, et ainsi de suite. Le problème de machines parallèles à traitement par batches, intervalles de durées d'exécution compatibles et dates de disponibilité généralisées est noté $Bm_2|G_p^U = INT, b = k, \bar{r}_i|C_{max}$. Étant donné que dans notre modèle les dates de disponibilité généralisées correspondent aux dates de fin d'exécution des tâches sur le premier étage, les dates de disponibilité sont de la forme $\bar{r}_i = i \times p$, où p désigne la durée d'exécution des tâches sur le premier étage, et le nombre de tâches disponibles à la date $\bar{r}_i$ est égal à $i \times m_1$ avec $1 \leqslant i \leqslant \lceil \frac{n}{m_1} \rceil$.

Comme le problème $Bm_2|G_p^U = INT, b = k, \bar{r}_i|C_{max}$ est NP-difficile même en l'absence de dates de disponibilité généralisées, nous développons dans cette section une méthode approchée de type PTAS. Dans la sous-section 3.2.1 nous présentons le principe du schéma d'approximation développé ici, ensuite les sous-sections 3.2.2, 3.2.3 et 3.2.4 présentent les différentes étapes de la méthode qui est explicitée dans la sous-section 3.2.5. Notons que dans la suite de cette section nous employons le terme *date de disponibilité* à la place de *date de disponibilité généralisée.*

3.2.1 Principe

L'objectif d'un schéma d'approximation polynomial (PTAS) est de fournir une solution de valeur inférieure à $1 + O(\epsilon)$ fois l'optimum en un temps polynomial par rapport à la taille de

l'instance, avec $0 < \epsilon < 1$.

À ces fins, le schéma d'approximation utilisé consiste, de manière générale, à effectuer plusieurs opérations d'arrondi et de partitionnement de l'horizon temporel afin de pouvoir limiter le nombre de durées d'exécution et de dates de disponibilité distinctes, et ainsi énumérer tous les ordonnancement réalisables. Ces opérations inspirées des travaux de Hall et al. [96], et détaillées dans la sous-section 3.2.2 composent la première phase de la méthode. Cette phase de transformation des données précède le partitionnement des tâches en deux groupes. Le premier groupe de tâches, composé des tâches courtes, est ordonnancé approximativement à l'aide d'une heuristique constructive développée dans la sous-section 3.2.3. Le second groupe, composé des tâches longues dont nous avons limité le nombre de durées d'exécution distinctes durant la phase de transformation des données, et est ordonnancé optimalement à l'aide d'un algorithme d'énumération présenté dans la sous-section 3.2.4.

Après cette introduction du schéma utilisé par le PTAS, nous indiquons ici le rapport entre ces phases et la garantie de performance du PTAS (c'est-à-dire obtenir une solution inférieure à $1 + O(\epsilon)$). La première transformation consiste à arrondir les durées d'exécution et les dates de disponibilité à la puissance entière inférieure de $(1 + \epsilon)$, cette transformation garantit un nombre restreint de dates de disponibilité et durées d'exécution distinctes, ce qui est utile pour énumérer l'ensemble des ordonnancements. Le partitionnement de l'horizon temporel permet quant-à-lui de diviser l'horizon temporel en intervalles géométriques, où un intervalle commence et se termine par une date de disponibilité, et au sein de chaque intervalle nous ordonnançons toutes les tâches disponibles mais non exécutées. Ceci nous permet de restreindre la résolution du problème à une résolution successive de sous-problèmes correspondant à ordonnancer (dans un intervalle) un ensemble de tâches sans dates de disponibilité. La perte (augmentation de la fonction objectif) entraînée par ces transformation est inférieure à $1 + O(\epsilon)$ fois la valeur de la solution optimale. Avec un nombre constant de transformations, la valeur de la fonction objectif reste inférieure à $1 + O(\epsilon)$ fois la valeur de l'optimum du problème initial.

3.2.2 Transformation des données

Cette sous-section est consacrée à la transformation des données, elle est décomposée en trois parties : dans la partie A, nous présentons les résultats permettant de partitionner l'horizon temporel, la partie B, présente quant-à-elle le partitionnement des tâches entre tâches courtes et tâches longues et un lemme caractérisant l'ordonnancement de chaque intervalle du découpage temporel ; enfin la partie C, présente la seconde transformation des données, portant sur les durées d'exécution des tâches longues.

A. Partitionnement de l'horizon temporel

Le premier lemme de cette partie nous permet de définir l'horizon temporel de l'ordonnancement, ensuite le lemme 3.2.2 permet en arrondissant les dates de disponibilité généralisées de partitionner cet horizon temporel.

Soit $\bar{r}_{max} = \lceil \frac{n}{m_1} \rceil p$ la plus grande date de disponibilité, et $a_{max} = \max_{j \in J}\{a_j\}$ la plus grande ordonnée des points initiaux des intervalles de durées d'exécution sur le second étage. Pour rappel $C_{max}^{FCBLPT}(J)$ désigne la charge minimale en regroupant les tâches dans des batches (voir section 3.1.2). Soit *opt* la valeur du makespan optimal. Nous obtenons alors le résultat suivant :

Lemme 3.2.1

$$\max\{\bar{r}_{max}, a_{max}, \frac{C_{max}^{FCBLPT}}{m_2}\} \leqslant opt \leqslant \bar{r}_{max} + a_{max} + \frac{C_{max}^{FCBLPT}}{m_2}$$

Preuve. Étant donné que $\bar{r}_{max}$, a_{max}, et $\frac{C_{max}^{FCBLPT}}{m_2}$ sont tous trois des bornes inférieures triviales du problème $Bm_2|G_p^U = INT, b = k, \bar{r}_i|C_{max}$, l'inéquation $\max\{\bar{r}_{max}, a_{max}, \frac{C_{max}^{FCBLPT}}{m_2}\} \leqslant opt$ est vérifiée. Pour démontrer que $v_l = \bar{r}_{max} + a_{max} + \frac{C_{max}^{FCBLPT}}{m_2}$ est une borne supérieure du problème, nous exhibons un ordonnancement dont la durée n'est pas plus grande que v_l, pour construire cet ordonnancement nous utilisons l'algorithme suivant :

1. Créer la liste de batches L_B en appliquant la règle FCBLPT.
2. Utiliser la règle FAM pour ordonnancer ces batches en commençant le premier batch à la date $\bar{r}_{max}$.

Dans ce cas la date de début d'exécution du dernier batch, noté B, est inférieure ou égale à $\bar{r}_{max} + \frac{C_{max}^{FCBLPT}}{m_2}$. La durée d'exécution du batch B est inférieure à a_{max}, la date de fin d'exécution du batch B est donc inférieure ou égale à $v_l = \bar{r}_{max} + a_{max} + \frac{C_{max}^{FCBLPT}}{m_2}$. Les bornes inférieure et supérieure définies par le lemme 3.2.1 sont donc valides.

□

Soit $M = \epsilon \times \max\{\bar{r}_{max}, a_{max}, \frac{C_{max}^{FCBLPT}}{m_2}\}$, avec $0 < \epsilon < 1$. D'après le lemme 3.2.1, la valeur de l'ordonnancement optimal est inférieure à $\frac{3}{\epsilon}M$.

La première transformation des données du problème concerne les dates de disponibilité. Cette transformation consiste à transformer chaque date de disponibilité généralisée $\bar{r}_i$ en un multiple de M, c'est-à-dire à arrondir chaque valeur de $\bar{r}_i$ au plus proche multiple de M. Soit $\tilde{r}_i = M\lfloor\frac{\bar{r}_i}{M}\rfloor$ la valeur arrondie de la date de disponibilité généralisée $\bar{r}_i$. Alors :

Lemme 3.2.2 *Avec une perte d'un facteur* $(1+\epsilon)$*, nous supposons qu'il existe au plus* $(\frac{1}{\epsilon}+1)$ *dates de disponibilité généralisées distinctes dans le problème initial.*

Preuve. Suite à la transformation des dates de disponibilité, l'écart entre une date de disponibilité originelle $(\bar{r}_i)$ et sa date de disponibilité modifiée $(\tilde{r}_i)$ est inférieur à M. Donc tout ordonnancement réalisable du problème avec dates de disponibilité arrondies fournit un ordonnancement réalisable du problème initial en retardant tous les batches d'au plus M unités. Étant donné que $M \leqslant \epsilon \times opt$, cette transformation entraîne une perte de $(1+\epsilon)$. De plus, $M \geqslant \epsilon \times \bar{r}_{max}$. En combinant ces deux inégalités nous obtenons que $\forall i \ \lfloor\frac{\bar{r}_i}{M}\rfloor \leqslant \frac{1}{\epsilon}$, d'où la présence d'au plus $(\frac{1}{\epsilon}+1)$ dates de disponibilité généralisées distinctes.

□

Ainsi transformé le problème contient $(\frac{1}{\epsilon}+1)$ dates de disponibilité distinctes qui sont des multiples de M. Nous partitionnons alors l'horizon temporel d'ordonnancement $[0, \frac{3}{\epsilon}M)$ en $(\frac{1}{\epsilon}+1)$ intervalles disjoints. Les $\frac{1}{\epsilon}$ premiers intervalles sont notés $\delta_i = [(i-1)M, iM)$ avec $i = 1, \ldots, \frac{1}{\epsilon}$ et le dernier intervalle est noté $\delta_{\frac{1}{\epsilon}+1} = [\frac{1}{\epsilon}M, \frac{3}{\epsilon}M)$. Notons par $\rho_1, \rho_2, \ldots, \rho_{\frac{1}{\epsilon}+1}$ avec $\rho_i = (i-1)M, \ \forall i = 1, \ldots, \frac{1}{\epsilon}+1$, les nouvelles dates de disponibilité généralisées.

B. Partitionnement des tâches

Suite à ce partitionnement de l'horizon temporel nous partitionnons les tâches en deux groupes, les tâches courtes et les tâches longues. Une tâche est dite *courte* si l'ordonnée du point initial de son intervalle de durées d'exécution est inférieure à ϵM. Sinon, la tâche est dite *longue*. Un batch est dit *long* s'il contient au moins une tâche longue, sinon il est dit *court*.

Le découpage de l'intervalle temporel accompagné du partitionnement des tâches permet à l'ordonnancement de chaque intervalle de respecter un certain nombre de règles définies dans le lemme suivant :

Lemme 3.2.3 *Il existe un ordonnancement optimal respectant les propriétés suivantes :*

1. *Sur chaque machine, les batches commençants dans le même intervalle δ_i sont exécutés par ordre décroissant de leur durée d'exécution.*
2. *Dans chaque intervalle δ_1 à $\delta_{\frac{1}{\epsilon}+1}$, les tâches sont affectées aux batches par ordre décroissant de la durée d'exécution des batches, de telle sorte que chaque batch contient les k plus longues tâches (ou le plus de tâches possible) compatibles avec le batch dans la limite du nombre de tâches disponibles.*
3. *Dans chaque intervalle δ_1 à $\delta_{\frac{1}{\epsilon}+1}$, au plus un batch long contient des tâches courtes.*

Preuve. Démontrons cette preuve propriété après propriété :

1. Supposons que les batches d'une machine m de l'intervalle δ_i ne soient pas exécutés par ordre décroissant de leur durée d'exécution. Changer l'ordre des batches et les séquencer dans l'ordre décroissant de leur durée d'exécution est possible car toutes les tâches sont disponibles en début d'ordonnancement, de plus cette transformation n'entraîne pas d'augmentation de la valeur du makespan.
2. En présence de compatibilité de type intervalles de durées d'exécution, remplir les batches d'un intervalle par ordre décroissant de leur durée d'exécution permet d'inclure plus de tâches dans les mêmes batches. De plus remplir les batches avec les k plus grandes tâches compatibles permet de réduire la somme des durées d'exécution des batches de l'ordonnancement. Ces deux affirmations peuvent être démontrées par l'argument d'échange de tâches.
3. Cette dernière propriété est induite par les deux propriétés précédentes.

□

C. Transformation des durées d'exécution

L'ordonnancement des tâches longues est effectué par énumération de tous les ordonnancements réalisables de tâches longues. Afin de limiter le nombre d'ordonnancements réalisables, nous utilisons la technique introduite par Afrati et al. [2] pour obtenir un nombre de durées d'exécution distinctes indépendant du nombre de tâches. Nous arrondissons toutes les ordonnées des points initiaux des intervalles de durées d'exécution en les multipliant par $(1+\epsilon)$ puis en les faisant décroître à la plus proche puissance entière de $(1+\epsilon)$.

Lemme 3.2.4 *Avec une perte d'un facteur $(1+\epsilon)$, nous supposons que les ordonnées des points initiaux des intervalles de durées d'exécution des tâches longues sont des puissances entières de $(1+\epsilon)$.*

Preuve. La transformation des ordonnées des points initiaux des intervalles de durées d'exécution présentée précédemment entraîne une perte inférieure à $(1+\epsilon)$, en effet la multiplication des durées d'exécution par $(1+\epsilon)$ entraîne une perte de $(1+\epsilon)$. Par la suite lorsque nous faisons décroître ces valeurs vers la plus proche puissance entière de $(1+\epsilon)$, les durées d'exécution modifiées restent supérieures aux durées d'exécution originelles. C'est pourquoi l'ordonnancement reste réalisable, et sa date de fin d'exécution est inférieure à $(1+\epsilon)$ fois celle de l'optimum.
□

Le lemme 3.2.5 nous permet d'obtenir une borne du nombre d'ordonnées de points initiaux des intervalles de durées d'exécution des tâches longues indépendante du nombre de tâches.

Lemme 3.2.5 *Le nombre d'ordonnées de points initiaux des intervalles de durées d'exécution des tâches longues, v, est borné par $\lfloor 1 + \log_{1+\epsilon} \frac{1}{\epsilon^2} \rfloor$.*

Preuve. Soit j la plus petite tâche longue, par définition $a_j \geqslant \epsilon M \geqslant \epsilon^2 a_{max}$. Or d'après le lemme 3.2.4, toutes les durées d'exécution des tâches longues sont des puissances entières de $(1+\epsilon)$. C'est pourquoi nous supposons qu'il existe un entier x tel que $a_j = (1+\epsilon)^x$ et que $a_{max} \geqslant (1+\epsilon)^{x+v-1}$. Alors $a_j = (1+\epsilon)^x \geqslant \epsilon^2(1+\epsilon)^{v-1}(1+\epsilon)^x$. D'où $v \leqslant \lfloor \log_{1+\epsilon} \frac{1}{\epsilon^2} + 1 \rfloor$.
□

Supposons les durées d'exécution des tâches longues distinctes notées $\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_v$. Nous gardons en mémoire les relations de compatibilité originelles entre les tâches.

Les différentes transformations effectuées, décrivons maintenant les algorithmes d'ordonnancement des tâches courtes et des tâches longues.

3.2.3 Ordonnancement des tâches courtes

Dans cette sous-section, nous supposons que toutes les tâches sont courtes. L'algorithme développé ici permet d'ordonnancer approximativement les tâches courtes dans les intervalles dont toutes les tâches disponibles n'ont pas été consommées. Dans le paragraphe suivant nous introduisons le principe de l'algorithme, avant de démontrer la perte engendrée par son application.

L'algorithme crée tout d'abord les batches à l'aide de la règle FCBLPT, soit L_B la liste de batches créée. Cette liste est triée par ordre décroissant du ratio $\frac{P(B_i)}{|B_i|}$, où $P(B_i)$ est la durée d'exécution du batch B_i, et $|B_i|$ son nombre de tâches. Sur la première machine disponible de l'intervalle δ_i, si un nombre suffisant de tâches est disponible pour exécuter le premier batch de la liste L_B, l'exécuter puis le supprimer de la liste L_B et réitérer cette opération. Sinon passer à l'intervalle suivant.

D'où :

Algorithme ST

1. Créer les batches en utilisant l'algorithme FCBLPT. Soit L_B la liste des batches créés.
2. Réindexer la liste L_B par ordre décroissant du ratio $\frac{P(B_i)}{|B_i|}$. Soit B le premier batch de la liste L_B, et $nb = 0$.
3. Pour chaque intervalle $\delta_i,\ i = 1, \ldots, \frac{1}{\epsilon} + 1$

(a) S'il n'y a pas assez de places[4] disponibles pour exécuter le batch B (c'est-à-dire $m_1\lfloor\frac{(i-1)M}{p}\rfloor - nb < |B|$) ou si aucune machine n'est disponible avant la fin de l'intervalle δ_i (impossible pour le dernier intervalle), alors aller à l'intervalle suivant δ_{i+1}.

(b) S'il y a assez de places disponibles pour exécuter le batch B (c'est-à-dire $m_1\lfloor\frac{(i-1)M}{p}\rfloor - nb \geqslant |B|$) et qu'une machine est disponible pour l'exécuter, ordonnancer le batch B sur la première machine disponible (règle FAM). Supprimer B de la liste L_B, ajouter $|B|$ à nb ($nb = nb + |B|$) et aller à 3a.

Lemme 3.2.6 *L'algorithme ST est un PTAS avec une perte d'au plus* $1 + (2 + \frac{1}{m_2})\epsilon^2$ *lorsque toutes les tâches sont courtes.*

Preuve. Soit S l'ordonnancement optimal et *opt* son makespan. Soit $\tilde{S}$ et $\tilde{opt}$, respectivement, l'ordonnancement obtenu par l'algorithme ST et son makespan. Considérons le dernier temps mort dans l'ordonnancement obtenu $\tilde{S}$, ce temps mort noté Δ se termine à la date t. Cette date t correspond à une date de disponibilité modifiée ρ_i, si ce n'est pas le cas le temps mort peut être retardé. Comme tous les batches de $\tilde{S}$ sont courts, tout batch commençant avant t est complété avant la date $t + \epsilon M$. Soit $\tilde{A}$ l'ensemble de batches de $\tilde{S}$ commençant après la date t. Étant donné que tous les batches de $\tilde{A}$ sont exécutés sans temps mort à partir de $t + \epsilon M$, et que $\tilde{S}$ est obtenu en appliquant la règle FAM, une borne supérieure de $\tilde{opt}$ est :

$$\tilde{opt} \leqslant t + \epsilon M + \frac{\sum_{B\in\tilde{A}} P(B)}{m_2} + P(B^{max}) \leqslant t + \frac{\sum_{B\in\tilde{A}} P(B)}{m_2} + 2\epsilon M \tag{3.1}$$

où $P(B^{max})$ désigne la plus grande durée d'exécution des batches de l'ensemble $\tilde{A}$.

Soit $\tilde{B}$ le batch reporté de l'intervalle δ_i vers l'intervalle δ_{i+1}, alors $\sum_{B\in\tilde{A}} P(B) = \sum_{B\in\tilde{A}\setminus\tilde{B}} P(B) + P(\tilde{B}) \leqslant \sum_{B\in\tilde{A}\setminus\tilde{B}} P(B) + \epsilon M$, où $\tilde{A}\setminus\tilde{B}$ représente l'ensemble des batches à ordonnancer après t.

Soit A la liste de batches ordonnancer après la date t dans l'ordonnancement optimal S. En observant les opérations de construction et d'indexation de l'algorithme ST, il est évident que $\sum_{B\in\tilde{A}\setminus\tilde{B}} P(B) \leqslant \sum_{B\in A} P(B)$. D'où $\sum_{B\in\tilde{A}} P(B) \leqslant \sum_{B\in A} P(B) + \epsilon M$ ce qui implique :

$$t + \frac{\sum_{B\in\tilde{A}} P(B)}{m_2} + 2\epsilon M \leqslant t + \frac{\sum_{B\in A} P(B) + \epsilon M}{m_2} + 2\epsilon M$$

De plus $t + \frac{\sum_{B\in A} P(B)}{m_2}$ est une borne inférieure de *opt*, donc en utilisant l'inégalité 3.1, nous obtenons $\tilde{opt} \leqslant opt + \frac{\epsilon M}{m_2} + 2\epsilon M$, or $M \leqslant \epsilon\ opt$ d'où :

$$\tilde{opt} \leqslant (1 + (2 + \frac{1}{m_2})\epsilon^2)opt$$

□

3.2.4 Ordonnancement des tâches longues

D'après le lemme 3.2.5 le nombre de durées d'exécution distinctes de tâches longues est supposé indépendant du nombre de tâches et donc constant par rapport à la taille de l'instance.

4. au sens du nombre de tâches à ordonnancer au début de l'intervalle δ_i

Ainsi nous pouvons énumérer tous les ordonnancements réalisables de tâches longues en un temps polynomial par rapport à la taille de l'instance originelle. Afin de faciliter l'énumération nous utilisons les concepts de *configuration machine* et de *profil d'exécution* développés par Hall et Shmoys [97]. Une *configuration machine* détaille pour une machine, intervalle après intervalle, le nombre de batches pour chaque durée d'exécution, alors qu'un *profil d'exécution* regroupe l'ensemble des configurations machines de l'ordonnancement, soit m_2 configurations.

Soit σ un ordonnancement réalisable. Supprimons de σ toutes les tâches et batches courts afin de conserver uniquement les batches longs vidés de leurs tâches, on les nomme *batches longs vides*. C'est-à-dire que seule la durée d'exécution du batch est conservée. Ainsi à chaque machine correspond une paire de vecteurs (λ, μ) qui respecte l'ordonnancement σ. Cette paire de vecteurs $\lambda = (\lambda_1, \ldots, \lambda_{\frac{1}{\epsilon}+1})$ et $\mu = (\mu_{1,1}, \ldots, \mu_{1,v}, \mu_{2,1}, \ldots, \mu_{2,v}, \ldots, \mu_{\frac{1}{\epsilon}+1,1}, \ldots, \mu_{\frac{1}{\epsilon}+1,v})$ est appelée *configuration de machine*, et λ et μ sont tels que λ_i désigne le nombre de batches longs exécutés sur la machine pendant l'intervalle δ_i, alors que $\mu_{i,j},\ \forall i = 1, \ldots, \frac{1}{\epsilon}, \forall j = 1, \ldots, v$ désigne le nombre de batches longs de durée d'exécution $\tilde{a}_j$ commençant dans l'intervalle δ_i sur la machine. Notons que $\forall i = 1, \ldots, \frac{1}{\epsilon},\ \sum_{i=1}^{v} \mu_{i,j} = \lambda_i$. La longueur de chaque intervalle δ_i, $i = 1, \ldots, \frac{1}{\epsilon}$ est égale à M, hormis le dernier intervalle $\delta_{\frac{1}{\epsilon}+1}$ dont la longueur est égale à $\frac{2}{\epsilon}M$. Or dans cette sous-section seuls les batches longs nous intéressent, donc la durée d'un batch est supérieure à ϵM. C'est pourquoi les $\frac{1}{\epsilon}$ premiers intervalles $\delta_i,\ i = 1, \ldots, \frac{1}{\epsilon}$ contiennent moins de $\frac{1}{\epsilon}$ batches longs, et l'intervalle $\delta_{\frac{1}{\epsilon}+1}$ contenant quant-à-lui au plus $\frac{2}{\epsilon^2}$ batches longs. De plus, les durées d'exécution des batches longs vides sont choisies parmi v valeurs, avec $v \leqslant \lfloor 1 + \log_{1+\epsilon} \frac{1}{\epsilon^2} \rfloor$ (lemme 3.2.5). Ainsi si w batches longs commencent sur la machine pendant l'intervalle δ_i, le nombre de configurations de cette machine possibles pour cet intervalle est au plus v^w. Comme w peut prendre chaque valeur entière entre 0 et $\frac{1}{\epsilon}$ le nombre total de configurations du vecteur $\mu_i = (\mu_{i,1}, \ldots, \mu_{i,v})$ est égal à $1 + v + \cdots + v^{\frac{1}{\epsilon}}$. Donc le nombre de configurations de machines pour tous les intervalles δ_i est inférieur à $(1 + v + \cdots + v^{\frac{1}{\epsilon}})^{\frac{1}{\epsilon}} \times (1 + v + \cdots + v^{\frac{2}{\epsilon^2}}) < 2^{\frac{1}{\epsilon}} \times v^{\frac{3}{\epsilon}}$, nombre indépendant de n.

Ce qui nous permet de définir un profil d'exécution comme un tuple $(m_1, \ldots, m_\psi)$, où m_i désigne le nombre de machines avec la configuration i dans l'ordonnancement. Ainsi nous considérons uniquement $(m+1)^\psi$ profils d'exécution, où $\psi < 2^{\frac{1}{\epsilon}} \times v^{\frac{3}{\epsilon}}$.

3.2.5 Schéma d'approximation polynomial (PTAS)

Les sous-sections précédentes nous ont permises de partitionner l'horizon temporel en $(\frac{1}{\epsilon}+1)$ intervalles, et de partitionner les tâches en deux groupes en indiquant comment traiter chaque groupe en l'absence de tâches de l'autre groupe. L'objectif de cette dernière sous-section est donc d'ordonnancer les deux groupes de tâches en harmonie. Nous allons tout d'abord présenter deux nouveaux lemmes afin de séparer les phases d'ordonnancement des deux groupes de tâches. Finalement nous présentons l'algorithme LT-ST qui est un schéma d'approximation polynomial (PTAS) pour le problème $Bm_2|G_p^U = INT, b = k, \bar{r}_i|C_{max}$.

Le principe de ce schéma d'approximation polynomial est le suivant : créer un profil d'exécution σ et assigner les tâches longues à ce profil (l'ordonnancement résultant doit être réalisable). L'objectif est ensuite d'ordonnancer les tâches courtes dans les intervalles δ_i $(i = 1, \ldots, \frac{1}{\epsilon} + 1)$ chaque fois qu'il est possible de placer le plus grand batch court, c'est-à-dire chaque fois qu'il y a assez de tâches disponibles au début de l'intervalle δ_i et qu'il y a un temps mort sur une machine pendant l'intervalle δ_i.

Lemme 3.2.7 *Avec une perte d'un facteur* $(1+\epsilon)$*, nous supposons qu'aucun batch court n'est exécuté dans deux intervalles différents.*

Preuve. Lorsque l'exécution d'un batch court dans un temps mort de l'intervalle δ_i se termine dans l'intervalle δ_{i+1}, nous étirons l'intervalle δ_i d'une longueur supplémentaire inférieure à ϵM, ce qui permet de compléter l'exécution du batch court durant l'intervalle δ_i. Comme au plus $\frac{1}{\epsilon}$ intervalles peuvent être étirés, la perte entraînée est inférieure à $\epsilon M \times \frac{1}{\epsilon}$. Or $M \leqslant \epsilon\ opt$, d'où une perte inférieure à $\epsilon\ opt$.

□

Lemme 3.2.8 *Avec une perte d'un facteur* $(1+\epsilon+\epsilon^2)$*, nous supposons qu'aucune tâche courte n'est ordonnancée dans un batch long.*

Preuve. D'après le lemme 3.2.3, dans chaque intervalle au plus un batch long peut contenir des tâches courtes. Or toutes les tâches courtes incluses dans un batch long sont compatibles entre elles, il est donc possible d'étendre l'intervalle d'au plus ϵM pour ordonnancer ces tâches courtes dans un batch court. Étant donné qu'il y a $(\frac{1}{\epsilon}+1)$ intervalles, et que $M \leqslant \epsilon\ opt$, la perte est inférieure à $(\epsilon+\epsilon^2)opt$.

□

Les concepts et lemmes présentés dans cette section nous permettent de définir l'algorithme permettant de résoudre le problème $Bm_2|G_p^U = INT, b = k, \bar{r}_i|C_{max}$.

Algorithme LT-ST

1. Créer tous les profils d'exécution possibles comme indiqué dans la sous-section 3.2.4
2. Pour chaque profil d'exécution, soit $nb = 0$.
   (a) Assigner une configuration à chaque machine, en accord avec le profil. Si ce n'est pas possible supprimer le profil.
   (b) Pour chaque machine et chaque intervalle, séquencer les batches longs vides le plus tôt possible dans l'ordre décroissant de leurs durées d'exécution. Si des batches doivent être reporté à l'intervalle suivant, supprimer le profil.
   (c) Pour chaque intervalle, remplir les batches longs vides de l'intervalle par ordre décroissant de leur durée d'exécution. Ce remplissage consiste à ce que chaque batch contienne les k (ou le plus possible) plus longues tâches compatibles dans la limite du nombre de tâches disponibles. Si un batch reste vide, supprimer le profil. Sinon, ajouter à nb le nombre de tâches ordonnancées, et mettre à jour le nombre de tâches restantes dans cet intervalle δ_i, c'est-à-dire $l_{\delta_i} = m_1 \lfloor \frac{(i-1)M}{p} \rfloor - nb$.
   (d) Exécuter l'algorithme ST pour ordonnancer les tâches courtes dans les temps morts laissés par les batches longs.
3. Sélectionner le meilleur ordonnancement réalisable.

Lemme 3.2.9 *L'algorithme LT-ST est un schéma d'approximation polynomial (PTAS) pour le problème* $Bm_2|G_p^U = INT, b = k, \bar{r}_i|C_{max}$*, avec une durée d'exécution en* $O(n \log n + n \times (m+1)^{\psi})$*, avec* $\psi < 2^{\frac{1}{\epsilon}} \times v^{\frac{3}{\epsilon}}$.

Preuve. Comme l'ordonnancement optimal est associé à l'un des $(m+1)^{\psi}$ profils d'exécution, et que tous les profils d'exécution sont explorés, l'algorithme LT-ST fournit une solution approchée de la solution optimale. L'algorithme LT-ST génère, pour un profil donné, un ordonnancement optimal des tâches longues. De plus l'application de l'algorithme ST entraîne une

perte inférieure à $1+(2+\frac{1}{m_2})\epsilon^2$. Cette perte, associée aux résultats des lemmes 3.2.2, 3.2.4, 3.2.7 et 3.2.8 nous obtenons que l'algorithme LT-ST entraîne une perte d'un facteur $1+4\epsilon+(3+\frac{1}{m_2})\epsilon^2$.

Notons que l'algorithme LT-ST exécute $(m+1)^\psi$ l'algorithme ST pour ordonnancer les tâches courtes. Or la première étape de l'algorithme ST est une opération de trie, qui peut être exécuter préalablement, la complexité des autres étapes de l'algorithme ST est $O(n)$, donc la complexité de l'algorithme LT-ST est $O(n \log n + n \times (m+1)^\psi)$.

□

3.3 Conclusion du chapitre

Dans la première section de ce chapitre nous avons développé 6 heuristiques constructives avec garantie de performance pour le problème de minimisation de la durée de l'ordonnancement d'un flowshop hybride avec machines à traitement par batches et compatibilité entre les tâches, problème noté $HF2(m_1, Bm_2)|G_p = INT, b = k|C_{max}$. Nous avons également adapté ces méthodes et leurs garanties de performance aux problèmes avec une seule machine sur l'un des étages ($HF2(1, Bm)|G_p = INT, b = k|C_{max}$ et $HF2(m, B)|G_p = INT, b = k|C_{max}$). À la lecture de la section présentant les résultats expérimentaux 3.1.5, nous remarquons qu'au moins une de ces heuristiques fournit très rapidement une solution intéressante. Effectivement pour nos instances de tests composées de 200 tâches les résultats étaient obtenues instantanément et l'écart moyen par rapport à la borne inférieure est inférieure à 1% alors que l'écart maximum est de 8%.

La seconde section de ce chapitre présente un schéma d'approximation polynomial (PTAS) pour le problème de minimisation de la durée de l'ordonnancement d'un flowshop hybride avec machines à traitement par batches, compatibilité entre les tâches et durées d'exécution identique sur le premier étage, problème noté $HF2(m_1, Bm_2)|G_p^U = INT, b = k, p_j = p|C_{max}$.

Nous avons publié les résultats présentés dans cette section dans la revue *Computer & Operation Research* [19] ainsi qu'en tant qu'article long lors de la conférence ROADEF'07. Dans ce manuscrit nous avons modifié la présentation des méthodes qui nous semble plus adapté à une thèse, nous avons également ajouté l'étude des méthodes pour le problème inverse.

Chapitre 4

Méthodes Exactes

Ce chapitre est dédié à la résolution exacte du problème de flowshop hybride avec machines à traitement par batches et intervalles de durées d'exécution compatibles ($HF2(m_1, Bm_2)|G_p = INT, b = k|C_{max}$). Alors que dans la première section du chapitre précédent nous avons défini des heuristiques constructives afin de fournir des solutions approchées du problème, les méthodes présentées dans ce chapitre permettent l'énumération implicite de toutes les solutions afin de fournir une solution optimale du problème $HF2(m_1, Bm_2)|G_p = INT, b = k|C_{max}$.

Sommaire

4.1 Méthodes par séparation évaluation pour le flowshop hybride . . . 78
4.1.1 Méthode de Carlier et Néron . . . 78
4.1.2 Adaptation au problème de flowshop hybride avec machines à traitement par batches au second étage . . . 79
4.2 Méthode directe . . . 80
4.2.1 Première étape . . . 80
4.2.2 Seconde étape. . . . 83
4.3 Méthode inverse . . . 85
4.3.1 Première étape . . . 85
4.3.2 Seconde étape. . . . 88
4.4 Résultats Expérimentaux . . . 90
4.4.1 Performances des heuristiques . . . 90
4.4.2 Comparaison des bornes inférieures . . . 91
4.4.3 Comparaison des deux méthodes exactes . . . 93
4.5 Conclusion du chapitre . . . 103

4.1 Méthodes par séparation évaluation pour le flowshop hybride

Les procédures par séparation évaluation (PSE) pour la minimisation du makespan sur un flowshop hybride classique (machines disjonctives) développées jusqu'à présent sont de deux types. Nous présentons leurs principes généraux dans ce paragraphe et détaillons la méthode de Carlier et Néron [31] dans la sous-section suivante. Le premier type de procédures, développé par Brah et Hunsucker en 1991 [26], consiste à ordonnancer les tâches étage après étage, du premier au dernier étage, et machine après machine. Pour cette énumération, il utilise un arbre de recherche avec deux types de nœuds où chaque type de nœud indique si la tâche est exécutée sur une nouvelle machine ou non (sur la même machine que la tâche précédente). Ce type de procédure a été amélioré par la suite notamment par Portmann et al. [184] et Vignier [215]. Le second type de procédure développé par Carlier et Néron en 2000 [31], ordonnance les tâches étage après étage, du plus critique au moins critique, à l'aide d'un ordonnancement de liste, contrairement à la méthode de Brah et Hunsucker qui ordonnance toutes les tâches d'une machine avant d'ordonnancer les tâches de la machine suivante.

Dans cette section nous présentons le principe de la méthode par séparation évaluation pour le flowshop hybride développée par Carlier et Néron [31] en 2000. Ensuite nous l'adaptons aux problèmes de minimisation du makespan pour des flowshops à deux étages.

4.1.1 Méthode de Carlier et Néron

Pour rappel, l'objectif d'une procédure par séparation évaluation (PSE) est d'énumérer implicitement toutes les solutions réalisables à l'aide d'un arbre de recherche pour lequel chaque nœud correspond à un sous-problème. La procédure développée par Carlier et Néron [31] énumère les ordonnancements réalisables à l'aide d'une liste de tâches pour chaque étage. Ces listes de tâches sont ordonnancées suivant la règle développée par Carlier [30]. Cette règle consiste à ordonnancer les tâches suivant la règle FAM (First-Available-Machine, voir 3.1.1) en respectant les contraintes de disponibilité de la tâche sur l'étage en cours d'ordonnancement. L'ordre de traitement des étages n'est pas chronologique mais la procédure commence par énumérer les ordonnancements de l'étage le plus critique pour terminer par ordonnancer l'étage le moins critique.

Chaque nœud de l'arbre de recherche correspond à un ensemble de listes de tâches, une liste complète (contenant n tâches) pour chaque étage totalement séquencé, et une liste partielle pour l'étage en cours de séquencement. Ainsi l'ordonnancement d'un étage ne peut donc pas être réalisé avant que tous les étages qui le précèdent soient séquencés. Par exemple, si le premier étage est l'étage le moins critique, aucun étage ne peut-être ordonnancé avant le dernier niveau de l'arbre. La limitation du nombre de nœuds explorés ne peut donc s'effectuer sur la base des ordonnancements, Carlier et Néron ont donc utilisé le concept de *fenêtre d'exécution*. Ce principe de fenêtre d'exécution utilise le fait que les opérations de la tâche qui précèdent et qui suivent l'étage courant doivent être exécutées avant et après l'exécution de l'opération courante, et ceci en moins de temps que pour le meilleur ordonnancement obtenu jusqu'à présent (durant le parcours de l'arbre ou à l'aide d'une heuristique). L'exécution des opérations précédant celle de l'étage actuel implique que la tâche actuelle ne peut commencer avant une certaine date. Mais si l'on souhaite terminer l'exécution des opérations qui suivent la tâche actuelle avant la meilleure date de fin actuelle, la tâche en cours d'ordonnancement doit être terminée à une certaine date. Ces deux propositions implique l'utilisation de fenêtres d'exécution.

4.1.2 Adaptation au problème de flowshop hybride avec machines à traitement par batches au second étage

Dans notre cas l'étage critique est soit le premier, soit le dernier. Or dans le chapitre précédent nous avons remarqué que les problèmes de minimisation du makespan pour les flowshops sont symétriques c'est pourquoi nous pouvons supposer que le premier étage est toujours l'étage le plus critique (du problème originel ou du problème inversé). Ainsi la méthode de Carlier et Néron consiste à énumérer tous les ordonnancements du premier étage, puis à résoudre le problème résultant au second étage.

En résumé, ces procédures par séparation évaluation pour le flowshop à deux étages sont divisées en deux phases. Lors de la première phase les ordonnancements du premier étage (du problème originel ou du problème inversé) sont énumérés, à la suite de cette énumération nous obtenons un ensemble d'ordonnancement réalisable du premier étage S. Lors de la seconde phase nous cherchons pour chaque ordonnancement réalisable du premier étage, l'ordonnancement du second étage minimisant le makespan. Ainsi à la fin de la seconde étape nous obtenons des ordonnancements réalisable du problème de flowshop hybride à deux étages. Les ordonnancements du premier étage sont utilisés par la seconde étape sous forme de dates de disponibilité des tâches, ainsi chaque ordonnancement est transformé en un ensemble de dates de disponibilité à l'aide de la transformation suivante : chaque date de fin d'exécution sur le premier étage est égale à la date de disponibilité de la tâche pour la seconde étape. Ce qui nous permet de résoudre deux problèmes de machines parallèles plutôt qu'un seul problème de flowshop hybride à deux étages. La première phase est une phase d'énumération des ordonnancements non dominés d'un problème de machines parallèles, alors que la deuxième étape consiste à résoudre un certain nombre de problème de machines parallèles avec dates de disponibilité. En fonction de l'étage le plus critique la méthode traite le problème originel ou le problème inverse, mais le principe reste inchangé. Cette méthode a été utilisée par Haouari et al. [101] pour le problème de minimisation de flowshop hybrides à deux étages avec machines disjonctives.

Lorsque nous adaptons cette technique à notre problème, nous remarquons que la différence de type de machines entre les deux étages impose de développer deux procédures distinctes. En effet dans le cas où le premier étage est plus critique, nous traitons le problème originel donc la première étape consiste à résoudre le problème d'énumération d'ordonnancements de machines parallèles $Pm_1||-$ et la seconde étape consiste à minimiser le makespan de machines parallèles à traitement par batches, intervalles de durées d'exécution compatibles et dates de disponibilité $Bm_2|G_p = INT, b = k, r_j|C_{max}$. En revanche lorsque l'étage le plus critique est le second étage la procédure s'applique au problème inverse $HF2(Bm_2, m_1)|G_p = INT, b = k|C_{max}$, ainsi la première étape est une énumération d'ordonnancements de machines parallèles à traitement par batches et intervalles de durées d'exécution compatibles $Bm_2|G_p = INT, b = k|-$ et la seconde étape consiste à minimiser le makespan de machines parallèles disjointes avec dates de disponibilité $Pm_1|r_j|C_{max}$. Nous appelons la première méthode, lorsque le premier étage est critique, *méthode directe*, et nous la présentons dans la section 4.2 alors que la seconde méthode, appelée *Méthode inverse*, est présentée dans la section 4.3.

La procédure peut être résumée comme suit :

1. Énumérer tous les ordonnancements du premier étage.
2. Pour chaque ordonnancement du premier étage :
   (a) Transformer les dates de fin d'exécution du premier étage en dates de disponibilité.
   (b) Résoudre le problème de machines parallèles avec dates de disponibilité résultant.

3. Retourner le meilleur ordonnancement obtenu.

Nous présentons les méthodes par étapes, en commençant par une brève introduction de l'objectif de l'étape et une introduction à la méthode implémentée pour atteindre cet objectif. Ensuite pour les étapes utilisant une procédure par séparation évaluation, nous détaillons le schéma de séparation, c'est-à-dire que nous définissons le critère de séparation entre les différents enfants d'un même parent. Enfin nous présentons les bornes inférieures et les différentes règles de dominance.

4.2 Méthode directe

Dans cette section nous développons une PSE pour le problème initial $(HF2(m_1, Bm_2)|G_p = INT, b = k|C_{max})$ à utiliser lorsque l'étage le plus critique est le premier étage. Comme nous l'avons indiqué dans la section précédente cette PSE est divisée en deux étapes. Dans la première étape, la procédure énumère tous les ordonnancements non dominés (pour le flowshop) du premier étage, c'est-à-dire l'énumération des solutions du problème $Pm_1||-$ où la valeur du champ γ indique l'énumération des solutions. La seconde étape de la procédure consiste à obtenir le meilleur ordonnancement du second étage pour chaque ordonnancement du premier étage. C'est-à-dire la résolution d'un problème de machines parallèles à traitement par batches et compatibilité entre les tâches $Bm_2|G_p = INT, b = k, r_j|C_{max}$ pour chaque ordonnancement du premier étage, représenté par les dates de disponibilité.

La figure 4.1 permet de visionner le déroulement de la PSE. L'arbre *principal* représente l'arbre créé lors de la première étape. Lorsqu'une feuille de cet arbre principal est créée (feuille représentée par un nœud carré), alors la seconde étape est exécutée pour obtenir le meilleur ordonnancement respectant l'ordonnancement du premier étage fourni par la feuille. Cette seconde étape développe ainsi un sous-arbre dit *secondaire* lors de la création de chaque feuille de l'arbre principal. Les deux étapes ne sont donc pas successives, mais la seconde étape est imbriquée dans la première.

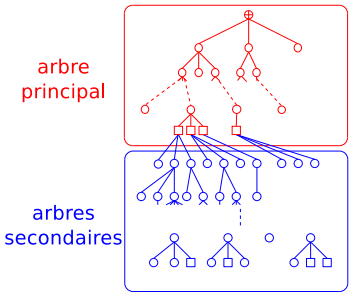


FIGURE 4.1 – Exemple d'arbre de recherche pour la méthode directe (PSE).

4.2.1 Première étape

L'objectif de cette étape est de fournir un ensemble d'ordonnancement du premier étage, tel qu'il existe un ordonnancement optimal du problème de flowshop hybride $HF2(m_1, Bm_2)|G_p = INT, b = k|C_{max}$ dont l'ordonnancement du premier étage appartient à l'ensemble d'ordonnancement fourni par cette étape. À défaut d'une méthode permettant d'obtenir directement un ordonnancement du premier étage fournissant un ordonnancement des deux étages optimal, nous énumérons tous les ordonnancements du premier étage pour lesquels nous n'avons pas la certitude qu'un autre ordonnancement du premier étage ne fournisse un ordonnancement global au moins aussi bon. Finalement chacun de ces ordonnancements est transformé en un ensemble de dates de disponibilité pour la seconde étape de la procédure.

Afin de simplifier la compréhension de la procédure, nous séparons l'arbre de recherche de la procédure globale (flowshop) en un arbre de recherche pour le premier étage, et un ensemble d'arbres de recherche pour le second étage tel que chaque feuille de l'arbre du premier étage corresponde à la racine de l'un des arbres du second étage. Il y a donc un arbre du second étage par feuille de l'arbre du premier étage, il est donc toujours possible de rassembler les différents arbres de recherche pour obtenir arbre de recherche originel.

Schéma de séparation. Lors de la première étape de la procédure, qui correspond au premier étage traité, un nœud représente une liste partielle de tâches. Un nœud N_l^1 de niveau l correspond à un ordonnancement partiel de l tâches, obtenu en ajoutant une tâche sur la première machine disponible suite à l'ordonnancement des $l-1$ tâches du nœud parent. Remarquons qu'étant donné que les tâches sont ajoutés en fin de liste, l'ordonnancement des l tâches du nœud N_l^1 n'est pas modifié par ses successeurs. Ainsi les nœuds de niveau n (et $n-1$ par déduction) correspondent à des ordonnancements complets du premier étage. Remarquons qu'il existe une bijection entre une liste et un ordonnancement, en effet, lorsque l'on impose que si plusieurs machines sont disponibles à la même date, la tâche doit être placée sur celle de plus petit indice, une liste de tâche fournit un ordonnancement unique. De plus, dans ces conditions, la manière de représenter un ordonnancement par une liste est unique.

Quant aux stratégies d'explorations de l'arbre de recherche, la présence des batches ne permet pas une utilisation simple des stratégies basées sur les divergences tels les méthodes développées par Hmida et al. [106]. En effet nous ne pouvons garantir l'optimalité d'une création de batches en temps polynomial pour le flowshop, et donc pour chaque divergence le nombre de possibilité serait trop important pour fournir des résultats intéressants. De plus deux ordonnancements du premier étage relativement différents peuvent fournir un ordonnancement global de même qualité. Il n'est donc pas intéressant d'énumérer tous les ordonnancements du premier étage avant de résoudre le premier problème du second étage. C'est pourquoi une stratégie d'exploration en "largeur d'abord" n'est pas adaptée à ce type de problème. C'est pourquoi nous avons préféré utiliser une stratégie de recherche en "profondeur d'abord" qui nous permet notamment d'améliorer rapidement notre borne supérieure.

La définition de l'arbre de recherche effectuée, nous remarquons que l'arbre possède autant de feuille qu'il existe de combinaison des n tâches, c'est à dire $n!$. Il est donc obligatoire, si l'on veut obtenir une méthode efficace de supprimer certaines branches de l'arbre, ce que nous permettent les bornes inférieures et les règles de dominances.

Bornes inférieures. Pour chaque nœud de l'arbre nous souhaitons savoir s'il est possible d'obtenir un ordonnancement du flowshop, basé sur la liste partielle de tâches du nœud en cours, de meilleure qualité que le meilleur ordonnancement connu. Si ce n'est pas possible alors il est inutile d'explorer la branche de ce nœud. Pour évaluer cette capacité nous calculons des bornes inférieures du makespan optimal d'ordonnancement des deux étages basés sur la liste de tâches partielle du nœud courant, puis nous comparons la plus grande de ces bornes avec le makespan du meilleur ordonnancement connu. Si la borne est supérieure au makespan du meilleur ordonnancement connu alors le nœud n'est pas exploré. Afin d'accélérer l'exécution de la PSE, nous utilisons comme meilleur ordonnancement initial la meilleure solution obtenue par les heuristiques présentées dans le chapitre Méthodes Approchées.

Soient $t_1^1, t_2^1, \ldots, t_{m_1}^1$ les dates de disponibilité des machines au premier étage, indexées par dates de disponibilité croissantes. Soient $\overline{J}$ l'ensemble des tâches non ordonnancées, et $p_1, p_2, \ldots, p_{|\overline{J}|}$ leurs durées d'exécution sur le premier étage, indexés par durées d'exécution croissantes, et $a_1, a_2, \ldots, a_{|\overline{J}|}$ les durées d'exécution minimales de ces tâches au second étage. Les plus petites dates de disponibilité de tâches au second étage sont notées $t_1^2, t_2^2, \ldots, t_{m_2}^2$ où $t_1^2 = t_1^1 + p_1$, $t_2^2 = \min\{t_2^1, t_1^2\} + p_2, \ldots$.

Borne[5] LBD_1^p**.** *Chaque tâche non ordonnancée $j \in \overline{J}$ doit être exécutée sur le premier puis sur le second étage.*

Aucune machine du premier étage n'est disponible avant la date t_1^1, donc la date de fin d'exécution d'une tâche j quelconque sur le second étage est supérieure à $t_1^1 + p_j + a_j$. D'où

$$LBD_1^p = \max_{j \in \overline{J}}(t_1^1 + p_j + a_j)$$

Bornes LBD_2^p **et** LBD_3^p**.** *Toutes les tâches non ordonnancées doivent être exécutées sur le second étage.*

Au moins une tâche doit être disponible pour que le premier batch puisse être exécuté, c'est pourquoi nous supposons que la date de disponibilité de la première machine égale à t_1^2. Au minimum une deuxième tâche doit être disponible pour exécuter la seconde tâche, nous supposons donc que la date de disponibilité de la seconde machine du deuxième étage égale à t_2^2...

Par ailleurs, nous relaxons le problème d'ordonnancement du premier étage en supposant que toutes les tâches sont disponibles au début de l'ordonnancement, et que les machines sont disponibles aux dates $t_1^2, \ldots, t_{m_2}^2$. Le problème résultant est un problème de machines parallèles à traitement par batches avec compatibilité entre les tâches, sans dates de disponibilité des tâches. Ce problème reste NP-Difficile, mais nous avons proposé une borne inférieure pour ce problème dans la section 3.1.2 égale à $\frac{C_{max}^{FCBLPT}(\overline{J})}{m_2}$ où $C_{max}^{FCBLPT}(\overline{J})$ désigne la charge minimale en regroupant les tâches de l'ensemble $\overline{J}$ dans des batches. D'où la borne :

$$LBD_2^p = \frac{\sum_{i=1}^{m_2} t_i^2 + C_{max}^{FCBLPT}(\overline{J})}{m_2}$$

Soit $J'_{r_j \geqslant t_1^1}$ la liste des tâches déjà placées sur le premier étage dont les dates de disponibilité au second étage sont supérieures à t_1^1. Ajoutons ces tâches aux tâches restantes dans la borne LBD_2^p. Alors, les dates de disponibilité $t_1'^2, t_2'^2, \ldots, t_{m_2}'^2$ des machines du second étage sont égales aux m_2 plus petites valeurs parmi $t_1^2, \ldots, t_{m_2}^2$ et r_j, $\forall j \in J'_{r_j \geqslant t_1^1}$. D'où :

$$LBD_3^p = \frac{\sum_{i=1}^{m_2} t_i'^2 + C_{max}^{FCBLPT}(\overline{J} \cup J'_{r_j \geqslant t_1^1})}{m_2}$$

Règles de dominances. Il est également possible d'éviter de créer certains nœuds en utilisant les règles de dominances. Ces règles permettent, sans le moindre calcul de ne pas explorer des nœuds dominés, qui ont un équivalent fournissant des ordonnancements de qualité égale. Nous présentons ici deux règles de dominance pour la première étape de la méthode directe.

5. La notation des bornes est la suivante LBT_i^{arbre}, où
   - T est égal à D si la borne est utilisée pour la méthode directe ou I si elle est utilisée pour la méthode inverse.
   - arbre est égal à p si la borne est utilisée dans l'arbre principal ou s si elle est utilisé dans l'arbre secondaire.

Première règle de dominance. Deux tâches sont *équivalentes* si leurs durées d'exécution sur les deux étages sont identiques.

Si deux tâches j_i et j_k sont équivalentes telles que $i < k$, alors la tâche j_i précède la tâche j_k.

Cette règle permet de ne pas explorer deux ordonnancements égaux pour lesquels seules deux tâches équivalentes sont échangées.

Seconde règle de dominance.

Les indices des m_1 premières tâches de la liste sont triés par ordre croissant.

Cette règle permet de ne pas explorer deux ordonnancements égaux pour lesquels les ensembles de tâches de deux machines ont été échangés.

4.2.2 Seconde étape.

L'objectif de cette étape est de fournir un ordonnancement optimal des deux étages dont l'ordonnancement du premier étage appartient à l'ensemble des ordonnancements du premier étage fourni par la première étape de la procédure. Étant donné que l'on ne peut toujours pas déterminer a priori un ordonnancement du premier étage fournissant un ordonnancement optimal, nous devons tester tous les ordonnancements du premier étage fourni par la procédure. Dans ces conditions, cette seconde étape doit chercher un ordonnancement optimal des deux étages pour chaque ordonnancement du premier étage. Les tâches sont disponibles pour être exécutées dans un batch au second étage dès que leur exécution est complétée au premier étage, c'est pourquoi nous transformons les dates de fin d'exécution au premier étage en dates de disponibilité au second étage. Suite à cette transformation la seconde étape consiste à résoudre pour chaque ordonnancement du premier étage un problème de machines parallèles à traitement par batches, compatibilité entre les tâches et dates de disponibilité ($Bm_2|G_p = INT, b = k, r_j|C_{max}$. Or ce problème reste NP-difficile, nous avons donc développé une procédure par séparation évaluation pour ce problème.

Schéma de séparation. En présence de batches à capacité finie, deux méthodes de branchement sont habituellement utilisées, la première consiste à ajouter, à chaque nœud, une tâche à la liste de tâches de son parent en indiquant si la tâche est la première d'un nouveau batch ou si elle est ordonnancée dans le même batch que la précédente. La seconde méthode consiste à ajouter, à chaque nœud, un batch à la liste des batches de son parent. Nous avons choisi la seconde méthode car elle permet, notamment, d'utiliser plus efficacement les incompatibilités et d'utiliser la dominance des solutions pour lesquels les batches sont remplis tant que leur dates de début et de fin ne sont pas modifiées.

La transformation d'une liste de batches en un ordonnancement du second étage s'effectue comme suit : lorsque toutes les tâches du premier batch sont disponibles, ce batch est placé sur la première machine disponible. Il est supprimé de la liste des batches et l'opération est réitérée tant qu'il reste des batches dans la liste.

Lors de cette seconde étape, au cours de laquelle l'ordonnancement du premier étage est fixe, un nœud représente une liste partielle de batches accompagnée de l'ordonnancement du premier étage. L'ordonnancement du premier étage étant commun à tout arbre de recherche du second étage, nous supposons par la suite que le nœud représente seulement la liste partielle de batches. Un nœud N_l^2 de niveau l, correspond à un ordonnancement partiel de l batches obtenu en ajoutant un batch sur la première machine disponible suite à l'ordonnancement des $l-1$ batches du nœud parent. Pour les mêmes raisons que pour le premier étage, l'ordonnancement

de ces L batches n'est pas remis en cause dans les successeurs de ce nœud. Cependant le nombre de tâches par batch n'étant pas fixe, les feuilles de l'arbre se situent à différents niveaux.

Comme la seconde étape représente des sous-arbres de l'arbre de recherche global, il est naturel de conserver une stratégie de recherche en "profondeur d'abord".

Bornes inférieures. Les bornes inférieures de cette seconde étape sont adaptées des deux types de bornes de la première étape. Dans cette étape l'ordonnancement du premier étage est fixé, ainsi les dates de disponibilité des tâches restantes sont données et non plus approximées. Supposons les tâches indexées par dates de disponibilité croissantes.

Borne LBD_1^s. *Chaque tâche non ordonnancée $j \in \overline{J}$ doit être exécutée sur une machine du second étage.*

Cette borne est une adaptation de la borne LBD_1^p. L'exécution des tâches non ordonnancées nécessite qu'une machine soit disponible et que la tâche le soit aussi. D'où la borne :

$$LBD_1^s = \max_{j \in \overline{J}}(\max\{t_1^2, r_j\} + a_j)$$

Borne LBD_2^s. *Toutes les tâches non ordonnancées doivent être exécutées sur le second étage.*

Cette seconde borne est une adaptation des bornes LBD_2^p et LBD_3^p. Au moins une tâche doit être disponible pour que le premier batch puisse être exécuté, c'est pourquoi nous supposons la date de disponibilité de la première machine disponible égale à t_1^2. Au minimum une deuxième tâche doit être disponible pour exécuter le second batch, nous supposons donc la date de disponibilité de la seconde machine du deuxième étage égale à t_2^2...

Comme pour la première étape, la borne inférieure de la charge moyenne minimale des machines du second étage que nous utilisons est égale à $\frac{C_{max}^{FCBLPT}(\overline{J})}{m_2}$. D'où la borne :

$$LBD_2^s = \frac{\sum_{i=1}^{m_2} \max\{t_i^2, r_i\} + C_{max}^{FCBLPT}(\overline{J})}{m_2}$$

Règles de dominance. La première règle que nous présentons est une adaptation de la première règle de la première étape. La seconde règle de la première étape étant spécifique aux premiers étages, elle n'est pas adaptée ici, cependant la seconde règle de cette étape est une règle spécifique aux derniers étages. Enfin la troisième règle est une règle spécifique aux batches, autrement dit les batches doivent être pleins s'il existe des tâches disponibles et compatibles avec la durée d'exécution du batch.

Première règle de dominance. Deux tâches sont *équivalentes* au second étage si leur intervalle de durée d'exécution est identique, et si leurs dates de disponibilité sont inférieures à la date de disponibilité de la première machine disponible, ou si leurs dates de disponibilité sont égales.

Si deux tâches j_i et j_k sont équivalentes telles que $i < k$, alors la tâche j_i précède la tâche j_k.

Seconde règle de dominance. Soient t_k^2 la plus petite date de disponibilité d'une machine du second étage, et $Cb_j^{min} = \max\{t_k^2, r_j\} + a_j$ la plus petite date de fin d'exécution d'un batch candidat $\bar{B}$ composé d'une seule tâche non-ordonnancées j.

Les dates de disponibilité des batches ajoutés au nœud sont inférieures à Cb_j^{min}.

Cette règle permet de ne pas explorer les ordonnancements dans lesquels des batches peuvent être exécutés pendant les temps morts.

Troisième règle de dominance.

Si des tâches peuvent être ajoutées au batch B sans modifier sa date de fin d'exécution, alors B ne peut pas être ajouté au nœud courant.

Cette règle permet d'imposer le remplissage des batches avec les tâches compatibles qui sont disponibles.

4.3 Méthode inverse

Dans cette section nous développons une PSE pour le problème inverse $(HF2(Bm_2, m_1)|G_p = INT, b = k|C_{max})$ à utiliser lorsque l'étage le plus critique est le second étage. Comme nous l'avons indiqué dans la section introductive de ce chapitre, cette PSE est divisée en deux étapes. La procédure énumère tous les ordonnancements non dominés (pour le flowshop) du premier étage du problème inverse durant la première étape, c'est-à-dire l'énumération des solutions du problème $Bm_2|G_p = INT, b = k|-$ où la valeur du champ γ indique l'énumération des solutions. La seconde étape de la procédure consiste à obtenir le meilleur ordonnancement du second étage pour chaque ordonnancement du premier étage. C'est-à-dire la résolution d'un problème de machines parallèles disjonctives $Pm_1|r_j|C_{max}$ pour chaque ordonnancement du premier étage, représenté par les dates de disponibilité.

Pour rappel l'ordonnancement ainsi obtenu est un ordonnancement optimal du problème initial dans lequel les tâches sont ordonnancées au plus tard. Toute cette section est consacrée au problème inverse, donc le premier étage désigne l'étage composé des machines à traitement par batches et le second étage celui composé des machines disjonctives. Afin de simplifier la compréhension nous inversons également les notations des étages, ainsi m_1 désigne le nombre de machines à traitement par batches et m_2 le nombre de machines disjonctives. Le problème à résoudre est donc désormais noté $HF2(Bm_1, m_2)|G_p = INT, b = k|C_{max}$, et les problèmes à résoudre au cours des deux étapes sont notés $Bm_1|G_p = INT, b = k|-$ et $Pm_2|r_j|C_{max}$.

4.3.1 Première étape

L'objectif de cette étape est de fournir un ensemble d'ordonnancement du premier étage, tel qu'il existe un ordonnancement optimal du problème de flowshop hybride $HF2(Bm_1, m_2)|G_p = INT, b = k|C_{max}$ dont l'ordonnancement du premier étage appartient à l'ensemble d'ordonnancement fourni par cette étape.

Le problème à résoudre au second étage peut ici être résolu à l'aide d'un algorithme de programmation dynamique. Afin de comparer les résultats obtenus par une méthode de programmation dynamique et ceux obtenus par une procédure par séparation évaluation pour le problème du second étage nous présentons ces deux types de méthodes. Lorsque l'on utilise une procédure par séparation évaluation au second étage, il est comme pour la méthode inverse intéressant de séparer l'arbre de recherche afin d'obtenir une méthode unifiée pour la première étape.

Schéma de séparation. Pour les mêmes raisons que lors du choix de la méthode de branchement de la seconde étape de la méthode directe, nous utilisons la méthode de branchement ajoutant un batch à la liste de batches du nœud parent.

La transformation d'une liste de batches en un ordonnancement du second étage s'effectue comme suit : lorsque toutes les tâches du premier batch sont disponibles, ce batch est placé sur la première machine disponible. Il est supprimé de la liste des batches et l'opération est réitérée tant qu'il reste des batches dans la liste.

Lors de cette seconde étape, au cours de laquelle l'ordonnancement du premier étage est fixe, un nœud représente une liste partielle de batches accompagnée de l'ordonnancement du premier étage. L'ordonnancement du premier étage étant commun à tout l'arbre de recherche du second étage, nous supposons par la suite que le nœud représente seulement la liste partielle de batches. Un nœud N_l^1 de niveau l, correspond à un ordonnancement partiel de l batches obtenu en ajoutant un batch sur la première machine disponible suite à l'ordonnancement des $l-1$ batches du nœud parent. Pour les mêmes raisons que pour le premier étage, l'ordonnancement de ces l batches n'est pas remis en cause dans les successeurs de ce nœud. Cependant le nombre de tâches par batch n'étant pas fixe, les feuilles de l'arbre se situent à différents niveaux.

Les caractéristiques du problème direct qui ont guidé notre choix de stratégie de recherche sont présentes également pour le problème inverse, c'est pourquoi nous utilisons également ici une stratégie de recherche en "profondeur d'abord". Pour rappel, la présence des batches ne permet pas une utilisation simple des stratégies basées sur les divergences tels les méthodes développées par Hmida et al. [106]. En effet nous ne pouvons garantir l'optimalité d'une création de batches en temps polynomial pour le flowshop, et donc pour chaque divergence le nombre de possibilité serait trop important pour fournir des résultats intéressants. De plus deux ordonnancements du premier étage relativement différents peuvent fournir un ordonnancement global de même qualité. Il n'est donc pas intéressant d'énumérer tous les ordonnancements du premier étage avant de résoudre le premier problème du second étage, une stratégie d'exploration en "largeur d'abord" n'est donc pas adaptée à ce type de problème. C'est pourquoi nous avons préféré utiliser une stratégie de recherche en "profondeur d'abord" qui nous permet notamment d'améliorer rapidement notre borne supérieure.

La définition de l'arbre de recherche effectuée, nous remarquons que l'arbre possède autant de feuille qu'il existe de combinaison des n tâches, c'est à dire $n!$. Il est donc obligatoire, si l'on veut obtenir une méthode efficace de supprimer certaines branches de l'arbre, ce que nous permettent les bornes inférieures et les règles de dominances.

Bornes inférieures. L'intérêt de l'utilisation d'une bonne inférieure ayant été développé pour la méthode directe, nous ne le rappelons pas ici.

Soient $t_1^1, t_2^1, \ldots, t_{m_1}^1$ les dates de disponibilité des machines au premier étage, indexées par dates de disponibilités croissantes. Soient $\overline{J}$ l'ensemble des tâches non ordonnancées, $a_1, a_2, \ldots, a_{|\overline{J}|}$ les durées d'exécution minimales sur le premier étage des tâches non ordonnancées $\overline{J}$ indexés par durées d'exécution croissantes, et $p_1, p_2, \ldots, p_{|\overline{J}|}$ représentent les durées d'exécution de ces tâches au second étage. Les plus petites dates de disponibilité de tâches au second étage sont notées $t_1^2, t_2^2, \ldots, t_{m_2}^2$ où $t_1^2 = t_1^1 + a_1, \ldots, t_k^2 = t_1^1 + a_k, t_{k+1}^2 = t_2^1 + a_{k+1} \ldots$

Borne LBI_1^p. *Chaque tâche non ordonnancée $j \in \overline{J}$ doit être exécutée sur le premier puis sur le second étage.*

Cette borne est une adaptation de la borne LBD_1^p. Aucune machine du premier étage n'est disponible avant la date t_1^1, donc la date de fin d'exécution d'une tâche j quelconque sur le second étage est supérieure à $t_1^1 + a_j + p_j$. D'où la borne :

$$LBI_1^p = \max_{j \in \overline{J}}(t_1^1 + a_j + p_j)$$

Bornes LBI_2^p et LBI_3^p. *Toutes les tâches non ordonnancées doivent être exécutées sur le second étage.*

Ces bornes sont des adaptations des bornes LBD_2^p et LBD_3^p. Pour que la première tâche puisse être exécutée sur le second étage, au moins un batch composé d'une seule tâche doit être compléter, c'est pourquoi nous supposons que la date de disponibilité de la première machine égale à t_1^2. Pour que la seconde tâche puisse être exécutée, une deuxième tâche doit être exécutée, si la capacité des batches le permet cette tâche peut appartenir au même batch que la première tâche. C'est pourquoi nous supposons que la date de disponibilité de la seconde machine disponible du deuxième étage est égale à t_2^2...

Par ailleurs, nous relaxons le problème d'ordonnancement du premier étage en supposant que toutes les tâches sont disponibles au début de l'ordonnancement, et que les machines sont disponibles aux dates $t_1^2, \ldots, t_{m_2}^2$, c'est-à-dire le problème $Pm_2||C_{max}$. Ce problème reste NP-Difficile, mais une borne habituelle de ce problème est une somme de la moyenne des dates de disponibilité des machines ($\sum t_i^2/m_2$), et de la durée totale d'exécution des tâches divisée par le nombre de machine ($\frac{\sum_{j \in \overline{J}} p_j}{m_2}$). D'où la borne :

$$LBI_2^p = \frac{\sum_{i=1}^{m_2} t_i^2 + \sum_{j \in \overline{J}} p_j}{m_2}$$

Soit $J'_{r_j \geqslant t_1^1}$ la liste des tâches déjà placées sur le premier étage dont les dates de disponibilité au second étage est supérieure à t_1^1. Ajoutons ces tâches aux tâches restantes dans la borne LBI_2^p. Alors, les dates de disponibilité, $t_1'^2, t_2'^2, \ldots, t_{m_2}'^2$ sont désormais les m_2 plus petites valeurs entre $t_1^2, \ldots, t_{m_2}^2$ et r_j, $\forall j \in J'_{r_j \geqslant t_1^1}$. D'où la borne :

$$LBI_3^p = \frac{\sum_{i=1}^{m_2} t_i'^2 + \sum_{j \in (\overline{J} \cup J'_{r_j \geqslant t_1^1})} p_j}{m_2}$$

Borne LBI_4^p. *Toutes les tâches non ordonnancées doivent être exécutées sur le premier étage.*

Aucun équivalent de cette borne n'est utilisé dans la méthode directe, en effet la durée totale d'exécution des batches n'est pas constante, ce qui est le cas pour la durée totale d'exécution des tâches. Toutes les tâches doivent être ordonnancées dans des batches au premier étage, les machines du premier étage étant disponibles aux dates $t_1^1, \ldots, t_{m_1}^1$. La moyenne des dates de fin d'exécution des derniers batches de chaque machine sur le premier étage est donc supérieure à la charge moyenne minimale des machines à traitement par batches pour exécuter les tâches non ordonnancées notée $\frac{C_{max}^{FCBLPT}(\overline{J})}{m_1}$ (voir la sous-section 3.1.2). Chacun de ces batches est composé d'au moins une tâche donc suite à l'exécution de ces batches au minimum m_1 tâches doivent être exécutées sur le second étage. D'où la borne :

$$LBI_4^p = \frac{\sum_{i=1}^{m_1} t_i^1 + C_{max}^{FCBLPT}(\overline{J})}{m_1} + \frac{\sum_{i=1}^{m_1} p_i}{m_2}$$

Règles de dominances. Les règles présentées pour cette étapes, sont adaptées de 3 des règles de la méthode directe. La première règle concernant l'ordre des tâches équivalentes est conservée, nous adaptons également la règle spécifique aux énumérations d'ordonnancements du premier étage empêchant l'échange de machines ainsi que la règle imposant le remplissage des batches sous certaines conditions.

Première règle de dominance. Deux tâches sont *équivalentes* si leurs durées d'exécution sur les deux étages sont identiques.

Si deux tâches j_i et j_k sont équivalentes telles que $i < k$, alors la tâche j_i précède la tâche j_k.

Seconde règle de dominance.

Les indices des premières tâches des m_1 premiers batches de la liste sont croissants.

Troisième règle de dominance. Soient t_1^1 et t_2^1 les dates de disponibilité des deux premières machines du premier étage, et B un batch non plein à ajouter sur la première machine disponible.

S'il existe une tâche non ordonnancée j telle que $a_j \leqslant p(B) \leqslant (1+\alpha)a_j$ et $t_1+p(B) > t_2+a_j$ alors B ne peut pas être ajouté.

Cette règle impose le remplissage des batches avec les tâches compatibles dont la date de fin d'exécution ne peut être inférieure si elles appartiennent au batch suivant, batch ordonnancé sur la seconde machine disponible.

4.3.2 Seconde étape.

L'objectif de cette étape est de fournir un ordonnancement optimal en utilisant les ordonnancements de la première étape. Étant donné que l'on ne peut pas déterminer a priori un ordonnancement du premier étage fournissant un ordonnancement optimal, nous devons tester tous les ordonnancements du premier étage fourni par la procédure. Dans ces conditions, cette seconde étape doit chercher un ordonnancement optimal des deux étages pour chaque ordonnancement du premier étage. Une tâche est disponible pour être exécutée au second étage dès que l'exécution du batch la contenant est complétée au premier étage, c'est pourquoi nous transformons les dates de fin d'exécution au premier étage en dates de disponibilité au second étage. Suite à cette transformation la seconde étape consiste à résoudre pour chaque ordonnancement du premier étage un problème de machines parallèles disjonctives avec dates de disponibilité ($Pm_2|r_j|C_{max}$). Or ce problème est NP-difficile au sens ordinaire d'après Lawler et al. [131], nous avons donc développé un algorithme de programmation dynamique. Nous avons également comparé cette méthode à la procédure par séparation évaluation développée par Gharbi et Haouari [84].

A. Algorithme de programmation dynamique.

Cette méthode de résolution est totalement différente des procédures par séparation évaluation que nous avons utilisées précédemment. C'est un algorithme de programmation dynamique pseudo-polynomial. Le problème $P|r_i|C_{max}$ n'est pas NP-difficile au sens fort contrairement au problème de la seconde étape de la méthode directe ($Bm_2|G_p = INT, b = k, r_j|C_{max}$), en effet le problème à une machine à traitement par batches et dates de disponibilité sans contraintes de compatibilité ($B|r_j|C_{max}$) est NP-difficile au sens fort (Brucker et al. [28]). Le fait que ce

problème soit NP-difficile au sens faible nous permet d'utiliser un algorithme de programmation dynamique pseudo-polynomial.

Lemme 4.3.1 *Il existe un ordonnancement optimal dans lequel les tâches d'une machines sont séquencées par dates de disponibilité croissantes.*

Preuve. Soit σ un ordonnancement optimal dans lequel la tâche i précède la tâche j avec $r_i \geqslant r_j$, et ces deux tâches sont ordonnancées sur la même machine. Soit t_i la date de début d'exécution de la tâche i, notons que la tâche j est disponible avant t_i donc elle est exécutée sans temps mort à la suite de la tâche i. L'échange des deux tâches permet d'exécuter la tâche j avant la date t_i si la machine est libre plus tôt, sinon elle est exécutée à la date t_i. Si à la fin de l'exécution de la tâche j la tâche i n'est pas disponible, alors la date de début d'exécution de i dans le nouvel ordonnancement est égale à t_i (sa date de début d'exécution dans σ). Sinon la tâche i est exécutée à la suite de la tâche j, comme l'exécution de ces deux tâches débute plus tôt elle se termine également plus tôt. Le nouvel ordonnancement est donc optimal. □

Algorithme. Soit $F_j(t_1, \ldots, t_{m_2})$, la durée totale de l'ordonnancement exécutant les tâches $1, \ldots, j$ tel que la première machine disponible soit disponible à la date t_1, la seconde à la date t_2... Remarquons que $F_j(t_1, \ldots, t_{m_2}) = t_{m_2}$. Les ordonnancements de makespan supérieur à celui d'un ordonnancement connu ne sont pas intéressant, c'est pourquoi nous traitons uniquement les valeurs de $F_j(t_1, \ldots, t_{m_2})$ inférieures à la borne supérieure. L'initialisation s'effectue comme suit :

$$F_0(r_1, \ldots, r_1) = r_1$$

L'équation de récurrence est la suivante :

$$F_j(\max\{t_1, r_{j+1}\}, \ldots, \max\{t_{m_2}, r_{j+1}\}) =$$

$$\min_{i=1,\ldots,m_2} \max\{r_{j+1}, C_j, F_{i-1}(t_1, \ldots, t_j - p_i, \ldots, t_{m_2})\}$$

La solution optimale est égale à la plus petite valeur de $F_n(t_1, \ldots, t_{m_2})$. Nous mettons à jour la borne supérieure pendant l'algorithme, ainsi chaque nouvelle valeur de $F_n(t_1, \ldots, t_{m_2})$ représente un ordonnancement réalisable meilleur que les ordonnancements obtenus jusqu'à présent. Finalement l'ordonnancement optimal est obtenu par «backtracking».

L'algorithme de programmation dynamique permet de résoudre le problème $Pm_2|r_i|C_{max}$ avec une complexité en $O(nC^m)$, où C est une borne supérieure de la solution obtenue. L'algorithme est donc pseudo-polynomial.

B. Procédure par séparation évaluation.

Pour cette étape, nous avions tout d'abord développé une procédure par séparation évaluation calquée sur celles des autres étapes, en effet nous avions uniquement adapté les bornes et les règles de dominances. Nous avons présenté ces résultats, avec une différence importante par rapport à la méthode utilisant la programmation dynamique, lors de la conférence ROADEF'09 à Nancy. Anis Gharbi, étonné par cette grande différence d'efficacité, nous a proposé d'utiliser la procédure par séparation évaluation qu'il a développée avec Mohamed Haouari pour le problème de machines parallèles avec dates de disponibilité et temps de finition $Pm|r_j, q_j|C_{max}$ [84].

Le schéma de séparation est identique à celui de la première étape de la méthode directe, c'est-à-dire qu'un nœud correspond à une liste partielle de tâches, et implicitement à un ordonnancement du premier étage, commun à tout l'arbre. À chaque nœud (tâche ajoutée) est associé une date de début, une date de fin et une date de fin impérative. Cette date de fin impérative est obtenue en combinant la borne supérieure et les intervalles de l'horizon temporel durant lesquels aucune machine n'est disponible. Sa date de disponibilité est également mise à jour en tenant compte des intervalles de l'horizon temporel durant lesquels aucune machine n'est disponible. Comme la tâche doit être exécutée sans interruption, si l'intervalle entre cette date de disponibilité et la date de fin impérative est inférieur à 2 fois la durée d'exécution de la tâche, celle-ci sera obligatoirement exécuté sur une machine durant un certain intervalle de temps. Ce qui nous permet de mettre à jour le nombre de machines disponibles pour chaque intervalle. Et de vérifier la faisabilité de ce nouvel ordonnancement. Pour de plus amples détails sur cette méthode de résolution se rapporter aux travaux de Gharbi et Haouari [84].

Nous avons essayé d'adapter cette utilisation des fenêtres de temps à notre problème, seulement les batches ont grandement compliqué les choses car soit nous considérions une machine à traitement par batch comme k machines disjonctives et dans ce cas le nombre de nœuds explorés ne diminuait pas, soit nous considérions les batches comme tels et dans ce cas le nombre de combinaisons à explorer imposait un nombre trop important de calculs, et donc un fort ralentissement de la procédure.

4.4 Résultats Expérimentaux

Les procédures par séparation évaluation sont des méthodes applicables à de petites instances. Le terme petit étant très vague, nous avons souhaité avoir une idée de la qualité de ces méthodes appliquées à notre problème, et de l'ordre de grandeur de la taille des problèmes résolus en un temps raisonnable. Dans un premier temps, nous justifions les différents choix effectués dans l'utilisation des heuristiques et des bornes inférieures. Ensuite nous comparons les trois méthodes, c'est-à-dire la méthode directe, la méthode inverse utilisant la programmation dynamique et la méthode inverse utilisant la PSE de Gharbi et Haouari.

Dans les sous-sections de comparaison des heuristiques et des bornes inférieures, les durées d'exécution des tâches sur le premier étage sont tirées aléatoirement dans l'intervalle $[5, 100]$ selon une distribution uniforme. Au second étage les intervalles de durées d'exécution sont proportionnels de la forme $[a_i, (1 + \alpha)a_i]$ où a_i est tiré aléatoirement dans l'intervalle $[5, 100]$ et α est égal à 0,1. La capacité des batches est égale à 2.

Pour toutes les sous-sections, en fonction des instances, il y a 2 ou 5 machines sur chaque étage. Les calculs sont effectués sur 5 instances pour chaque combinaison de machines pour l'analyse des bornes et sur 10 instances pour la comparaison des méthodes. Dans tous les tableaux de résultats, les durées sont exprimées en seconde, et les écarts relatifs en pourcentage.

4.4.1 Performances des heuristiques

Dans la sous-section 3.1.5, nous avons déjà présenté des résultats expérimentaux pour les heuristiques sur des instances de 200 tâches. Ces résultats expérimentaux portent sur des comparaisons avec les bornes inférieures. Dans cette section, nous avons effectué de nouvelles expérimentations sur les heuristiques, mais en comparant les résultats de ces heuristiques avec la solution optimale obtenue à l'aide de l'une des procédure par séparation évaluation. Pour obtenir une valeur optimale à coup sûr nous avons utilisé des instances de seulement 12 tâches. Le

calcul des heuristiques étant quasi instantané, même pour de grandes instances, nous comparons uniquement la qualité des solutions fournies par les heuristiques. Le critère retenu est l'écart relatif (en %) entre la date de fin de l'ordonnancement C^H_{max} fournie par l'heuristique H et la valeur optimale C_{opt} obtenue par l'une des PSEs : $\frac{C^H_{max}-C_{opt}}{C^H_{max}} \times 100$.

Le tableau 4.1 présente les écarts relatifs obtenus pour les heuristiques H_J, basée sur la règle de Johnson, H^{Inv}_J basée sur la règle de Johnson en inversant les étages, H_{LBPT} basée sur la règle $LBPT$ et enfin l'heuristique H^{Inv}_{LBPT} basée sur la règle $LBPT$ en inversant les étages. Pour chaque heuristique, la première valeur représente la moyenne des écarts relatifs, alors que la seconde représente le plus grand écart entre la solution obtenue et la solution optimale.

(m_1,m_2)	H_J		H^{Inv}_J		H_{LBPT}		H^{Inv}_{LBPT}	
	Moy	Max	Moy	Max	Moy	Max	Moy	Max
$(2,2)$	5,6	8,6	3,9	8,5	5,8	8,8	10,5	14,6
$(2,5)$	3,7	7,7	7,4	11,1	3,2	7,3	7,8	11,1
$(5,2)$	3,4	6,7	6,4	15,8	15,7	23,7	26,6	38,8
$(5,5)$	11,5	20,8	16,1	22,3	11,2	20,8	18,1	20,8
Total	6,1	20,8	8,5	22,3	9,0	20,8	14,0	38,8

TABLE 4.1 – Comparaison des heuristiques par rapport à l'optimum (PSE).

Nous observons que l'heuristique H_J domine globalement les trois autres heuristiques. Lorsqu'il y a 2 machines au premier étage et 5 au second, l'heuristique H_{LBPT} est meilleure, et lorsqu'il y a 2 machines sur chaque étage, l'heuristique H^{Inv}_J semble très légèrement la dominer. Malgré cette dominance de l'heuristique H_J, nous conservons les quatre heuristiques. En effet, nous avons remarqué que quelle que soit l'heuristique, il existe une instance pour laquelle l'heuristique domine les 3 autres. De plus la durée d'exécution de ces heuristiques est infinitésimale comparée à l'exécution de la PSE.

4.4.2 Comparaison des bornes inférieures

L'un des points critiques d'une PSE réside dans l'utilisation de bornes inférieures adaptées. La complexité de calcul des bornes étant identique, nous avons décidé de comparer les bornes sur leur seule efficacité. L'indicateur d'efficacité que nous utilisons ici est le temps de résolution. Cependant étant donné que pour tester une borne nous exécutons la méthode en supprimant toutes les autres bornes de l'étape (de la borne à tester), lorsque la borne n'est pas adaptée à l'instance nous n'obtenons pas de solution optimale, dans la limite de temps d'exécution retenue. Dans ce cas nous comparons les dates de fins d'ordonnancement des solutions obtenues.

Les instances de tests sont identiques pour toutes les bornes des méthodes directe et inverse, elles sont composées de 10 tâches. Pour chaque borne, la première valeur (*"Moy"*) représente la durée d'exécution moyenne (en secondes) pour les instances résolues, la seconde (*"Max"*) la plus grande durée d'exécution des instances résolues. La troisième valeur (*"Non Opt."*) indique le nombre d'instances non résolues, la dernière colonne (*"Non Best"*) indique le nombre d'instances dont la solution fournie après 10 minutes ne correspond pas à la meilleure solution obtenue par toutes les méthodes appliquées (directes et inverses).

A. Bornes inférieures de la méthode directe.

Le tableau 4.2 permet de comparer les 3 bornes que nous avons développées pour le premier étage de la méthode directe. Nous observons que la borne LBD_1^p surpasse toutes les autres bornes quel que soit le type d'instances. Pour les instances testées les performances lors de l'exécution de la borne LBD_1^p sont identiques à celles d'une exécution utilisant toutes les bornes. Il est donc intéressant d'observer les résultats obtenus sur des instances plus conséquentes, afin de déterminer s'il est préférable de supprimer les bornes LBD_2^p et LBD_3^p qui semblaient pourtant très utiles. Nous avons effectué ces expériences pour 15 tâches, présentées dans le tableau 4.3, et pour 20 tâches dans le tableau 4.4. Toutes les instances testées ne pouvant être résolues en 10 minutes, nous avons également inclus l'écart relatif par rapport à la meilleure solution obtenue. Ces tableaux indiquent qu'il n'est pas avantageux d'exécuter uniquement la borne LBD_1^p pour la première étape, en effet dans la majorité des cas elle fournit des résultats identiques, mais peut s'avérer moins bonne (durée moyenne lorsqu'il y a 15 tâches et respectivement 5 et 2 machines).

(m_1,m_2)	LBD_1^p			LBD_2^p				LBD_3^p				Toutes	
	Temps		Non	Temps		Non	Non	Temps		Non	Non	Temps	
	Moy	Max	Opt.	Moy	Max	Opt.	Best	Moy	Max	Opt.	Best	Moy	Max
$(2,2)$	2	5	0	46	182	1/5	0	77	183	0	0	1	4
$(2,5)$	2	5	0	263	442	1/5	0	-	-	5/5	0	3	6
$(5,2)$	13	33	0	15	42	0	0	14	50	0	0	10	33
$(5,5)$	0	0	0	16	81	0	0	31	156	0	0	0	0
Total	4	33	0	85	442	2/20	0	41	183	5/20	0	4	33

TABLE 4.2 – Comparaison des durées d'exécution pour chaque borne de la première étape de la méthode directe (PSE).

(m_1,m_2)	Directe Toutes bornes						Directe LBD_1^p seule					
	Temps		Non	Non	Écart		Temps		Non	Non	Écart	
	Moy	Max	Opt.	Best	Moy	Max	Moy	Max	Opt.	Best	Moy	Max
$(2,2)$	-	-	5/5	0	-	-	-	-	5/5	0	-	-
$(2,5)$	-	-	5/5	1/5	9,9	9,9	-	-	5/5	1/5	9,9	9,9
$(5,2)$	-	-	5/5	4/5	2,3	5,3	-	-	5/5	4/5	3,4	5,3
$(5,5)$	218	218	4/5	4/5	2,6	6,5	195	195	4/5	4/5	2,6	6,5
Total	218	218	19/20	9/20	3,3	9,9	195	195	19/20	9/20	3,8	9,9

TABLE 4.3 – Comparaison des durées d'exécution de la borne LBD_1^p pour 15 tâches (PSE).

Le tableau 4.5 permet quant-à-lui de comparer les 2 bornes que nous avons développées pour le second étage. Hormis lorsqu'il y a deux machines sur chaque étage, les résultats sont similaires quelle que soit la borne. Dans ce cas la borne LBD_2^s seule permet une résolution plus rapide que la borne LBD_1^s. Cependant cumuler les deux bornes permet de gagner quelques secondes supplémentaires.

(m_1,m_2)	Directe Toutes bornes						Directe LBD_1^p seule					
	Temps		Non	Non	Écart		Temps		Non	Non	Écart	
	Moy	Max	Opt.	Best	Moy	Max	Moy	Max	Opt.	Best	Moy	Max
$(2,2)$	-	-	5	1	5,0	5,0	-	-	5	1	5,0	5,0
$(2,5)$	-	-	5	0	-	-	-	-	5	0	-	-
$(5,2)$	-	-	5	2	3,4	3,5	-	-	5	2	3,4	3,5
$(5,5)$	70	70	4	2	1,5	2,0	70	70	4	2	1,5	2,0
Total	70	70	19/20	5/20	3,0	5,0	70	70	19/20	5/20	3,0	5,0

TABLE 4.4 – Comparaison des durées d'exécution de la borne LBD_1^p pour 20 tâches (PSE).

(m_1,m_2)	LBD_1^s				LBD_2^s			Toutes Bornes	
	Temps		Non	Non	Temps		Non	Temps	
	Moy	Max	Opt.	Best	Moy	Max	Opt.	Moy	Max
$(2,2)$	1	4	0	0	1	3	0	1	4
$(2,5)$	3	6	0	0	3	6	0	3	6
$(5,2)$	166	339	1/5	0	14	49	0	10	33
$(5,5)$	0	0	0	0	0	1	0	0	0
Total	42	339	1/20	0	5	49	0	4	33

TABLE 4.5 – Comparaison des durées d'exécution pour chaque borne de la seconde étape de la méthode directe (PSE).

B. Bornes inférieures de la méthode inverse.

Les résultats relatifs aux bornes de la méthode inverse sont présentés dans le tableau 4.6. Lorsqu'il y a 5 machines sur chaque étage, la résolution du problème est instantanée. Les autres cas sont plus intéressants, en effet lorsqu'il y a 5 machines au premier étage et 2 au second, seules les bornes LBI_1^p et LBI_3^p obtiennent la solution optimale. Lorsqu'il y a 2 machines au premier étage et 5 au second, la borne LBI_1^p ne vérifie aucune optimalité, alors que la borne LBI_2^p résout 3 instances en moins de 35 secondes pour les 2 autres instances elle obtient un ordonnancement optimal sans vérifier son optimalité. Enfin lorsqu'il y a 2 machines sur chaque étage, la borne LBI_2^p seule ne résout aucune instance, et la solution obtenue est moins bonne que celle obtenue par les autres bornes, qui résolvent dans le meilleur des cas une seule instance. Pour conclure, nous remarquons que lorsque nous utilisons toutes les bornes une seule instance n'est pas résolue, mais la solution fournie pour cette instance est moins bonne que les solutions obtenues lorsque LBI_2^p n'est pas utilisée.

4.4.3 Comparaison des deux méthodes exactes

Dans cette section, nous évaluons la méthode directe et les méthodes inverses (programmation dynamique et PSE de Gharbi et Haouari) sur plusieurs types d'instances. De petites instances, qui contiennent 12 tâches, ces instances sont toutes résolues. Des instances de taille raisonnable (15 tâches), pour lesquelles le critère retenu est le temps d'exécution de la méthode, et le nombre d'instances résolues sachant que la plupart des instances sont résolues optimalement par l'une des méthodes. Nous effectuons également les tests sur des instances plus conséquentes (20 tâches), dans ce cas nous comparons les dates de fin des meilleurs ordonnancements obtenus

(m_1, m_2)	LBI_1^p				LBI_2^p				LBI_3^p			
	Temps		Non	Non	Temps		Non	Non	Temps		Non	Non
	Moy	Max	Opt.	Best	Moy	Max	Opt.	Best	Moy	Max	Opt.	Best
$(2,2)$	276	276	$4/5$	0	172	309	$1/5$	$1/5$	368	368	$4/5$	0
$(2,5)$	-	-	$5/5$	0	12	34	$2/5$	0	-	-	$5/5$	$1/5$
$(5,2)$	0	4	0	0	198	442	$1/5$	0	72	195	0	0
$(5,5)$	0	3	0	0	2	10	0	0	1	6	0	0
Total	31	276	$9/20$	0	95	442	$4/20$	$1/20$	67	368	$9/20$	$1/20$

(m_1,m_2)	LBI_4^p				Toutes Bornes			
	Temps		Non	Non	Temps		Non	Non
	Moy	Max	Opt.	Best	Moy	Max	Opt.	Best
$(2,2)$	-	-	$5/5$	0	68	141	$1/5$	$1/5$
$(2,5)$	84	170	$1/5$	$1/5$	86	427	0	0
$(5,2)$	590	590	$4/5$	0	1	4	0	0
$(5,5)$	1	8	0	0	0	0	0	0
Total	93	590	$10/20$	$1/20$	35	427	$1/20$	$1/20$

TABLE 4.6 – Comparaison des durées d'exécution pour chaque borne de la méthode inverse (PSE).

après une durée d'exécution de 10 minutes. Les durées d'exécution sur le premier étage sont tirées aléatoirement dans l'intervalle $[15, 85]$ selon une distribution uniforme. Au second étage les intervalles de durées d'exécution sont proportionnels de la forme $[a_i, (1+\alpha)a_i]$, la valeur de α est égale à 0,1 ou 1 et les durées minimales d'exécution, a_j sont tirées aléatoirement dans l'un des intervalles $[15, 85]$, $[30, 170]$, $[45, 255]$ ou $[60, 340]$. C'est-à-dire dans les intervalles de la forme $[15c_2, 185c_2]$ avec $c_2 \in \{1, 2, 3, 4\}$, où c_2 indique le rapport moyen entre les durées d'exécution des deux étages. La capacité des batches est égale à 2 ou 3. Les combinaisons de nombre de machines sont notées $m_1 - m_2$, telles que m_1 représente le nombre de machines sur le premier étage, et m_2 le nombre de machines sur le second étage de type batch. Les combinaisons utilisées sont : 2-2, 2-5, 5-2, 5-5. Chaque combinaison de caractéristiques est testée sur 10 instances.

La première colonne des tableaux indique le nombre de machines sur le premier et le second étage (2 ou 5), la seconde indique la capacité des batches (2 ou 3), la troisième, l'indice de proportionnalité de la durée d'exécution maximale sur le second étage (0,1 ou 1), et la quatrième, le coefficient multiplicateur de l'intervalle de sélection des durées d'exécution du second étage ($c_2 = 1, 2, 3, 4$). La double colonne "Heuristiques" indique le gain relatif de la meilleure solution obtenue par les PSEs, notée C_{max}^{PSE} par rapport à la solution obtenue par l'heuristique H^*, notée $C_{max}^{H^*}$. La première colonne ("Écart Moy.") indique la moyenne des ratios $\frac{C_{max}^{H^*} - C_{max}^{PSE}}{C_{max}^{PSE}} \times 100$ et la seconde ("Écart Max") le plus grand des ratios obtenus parmi les 10 instances. Les colonnes suivantes présentent, par groupe de 6, les résultats de la méthode directe (*Directe (PSE)*), de la méthode inverse utilisant la programmation dynamique (*Inverse (dynamique)*) et de la méthode inverse utilisant la PSE pour les deux étapes (*Inverse (PSE)*). Les deux premières colonnes ("Temps Moy." et "Temps Max") de chaque groupe présentent, respectivement, les durées moyenne et maximale d'exécution de la PSE pour les instances résolues en moins de 10 minutes. La colonne "Non Opt." indique, parmi les 10 instances, le nombre d'instances n'ayant pas été résolues. La colonne "Non Best" indique le nombre de solutions qui sont dominées par la solution obtenue par une autre méthode. Les deux dernières colonnes ("Écart Moy." et "Écart

Max") présentent, respectivement, les écarts relatifs moyen et maximum entre la solution obtenue par l'heuristique, lorsque l'instance n'a pas été résolue, et la meilleure borne inférieure.

À la lecture des tableaux 4.7, 4.8, 4.9 et 4.10, nous remarquons que contrairement aux heuristiques, pour lesquelles les différentes valeurs de α fournissent des ordonnancements de même qualité, pour les méthodes exactes inverses le nombre d'instances résolues est plus important si la valeur de α est faible. Ceci s'explique par le nombre de combinaisons de batches qui croît avec la valeur de α, les nouveaux batches peuvent contenir des tâches disparates. En revanche nous ne constatons aucun écart significatif pour la méthode directe.

À l'instar des heuristiques, la taille des batches modifie la qualité des solutions obtenues par les méthodes inverses. Cependant dans la section 3.1.5, nous avons constaté que l'augmentation de la capacité entraîne une dégradation des solutions obtenues par les heuristiques. Au contraire, dans le cas des méthodes inverses, les instances avec une capacité de 3 sont plus souvent résolues que celles avec une capacité de 2. En effet, la plus grande capacité des batches permet de créer des ordonnancements avec moins de batches, d'où une profondeur de l'arbre de décision moindre. Contrairement aux méthodes inverses, l'efficacité de la méthode directe ne dépend pas des propriétés des batches (k et α).

Observons désormais les variations par rapport à c_2 : critère qui permet d'obtenir des durées d'exécution du second étage plus ou moins longues, ce qui modifie le rapport de charge entre les 2 étages. D'après les tableaux 4.7 et 4.10, nous remarquons que la méthode directe fournit généralement de meilleures solutions que les méthodes inverses lorsque $c_2 = 1$; en revanche pour les autres valeurs de c_2 elle fournit de mauvaises solutions. Alors que la différence de qualité des solutions obtenues par les heuristiques inverses est importante entre $c_2 = 1$ et $c_2 = 2$, il n'y a aucune différence significative entre les résultats concernant $c_2 = 2, 3$ et 4. Nous pouvons donc en déduire que lorsque le nombre de machines est le même sur chaque étage, le second étage devient critique dès que $c_2 = 2$.

Le tableau 4.8 confirme l'intérêt d'utiliser la méthode directe pour obtenir les meilleures solutions des instances pour lesquelles le premier étage est critique. En effet pour ces instances avec, respectivement, 2 et 5 machines et un coefficient $c_2 = 1, 2$, la méthode directe fournit toutes les meilleures solutions (sauf une) en montrant l'optimalité de seulement 2 solutions lorsqu'il y a 20 tâches. Elle obtient également la plupart des meilleures solutions lorsque $c_2 = 3, 4$. Notons cependant que la méthode inverse avec programmation dynamique, même si elle ne fournit jamais de meilleures solutions, résout la moitié des instances lorsque $\alpha = 0.1$ et $c_2 = 1, 4$. En revanche, alors que la méthode inverse avec programmation dynamique résout toutes les instances de 12 tâches, lorsqu'il y a 20 tâches aucune tendance significative n'est observée.

Lorsqu'il y a 5 machines sur le premier étage et 2 sur le second, le tableau 4.9 indique que le second étage est quasiment toujours critique ; donc la méthode directe ne fournit pas de résultats intéressants pour cette combinaison de nombre de machines. Une combinaison de paramètres k et α se détache clairement de ce tableau ; en effet, lorsque $k = 3$ et $\alpha = 0.1$, la majorité des instances de 20 tâches sont résolues, alors que seules 2 instances des autres combinaisons sont résolues. Cela s'explique par le fait qu'une petite valeur de α diminue le nombre de tâches compatibles, et donc diminue le nombre de combinaisons de batches réalisables. Parallèlement une grande valeur de k augmente le nombre de tâches par batch, et donc diminue le nombre de batches à placer. Ainsi les nombres de décision, et de choix pour chaque décision sont limités. Dans les autres cas nous remarquons que lorsque $c_2 = 3, 4$, les méthodes inverses fournissent toujours la meilleure solution (sauf une fois), alors que lorsque $c_2 = 1, 2$, les meilleures solutions sont partagées entre les méthodes exactes (avec quelques meilleures solutions obtenues uniquement par la méthode

directe).

La comparaison des méthodes en fonction du nombre de machines sur chaque étage confirme l'importance de l'équilibre entre les étages. Ainsi, nous remarquons que la méthode directe fournit plus souvent la meilleure solution lorsque le premier étage est plus critique (2-5 meilleur que 2-2 ou 5-5 alors que 5-2 est moins bon). Alors que les méthodes inverses fournissent de moins bons résultats lorsqu'il y a respectivement 2 et 5 machines, le cas où le premier étage est le plus critique. Enfin alors que la méthode utilisant la programmation dynamique fournit des résultats comparables pour les autres cas (2-2, 5-2 et 5-5), la méthode inverse utilisant la PSE pour les deux étapes fournit de meilleurs résultats lorsqu'il y a respectivement 5 et 2 machines.

Finalement, ces différents tableaux nous ont permis de constater que la qualité de résolution de ces méthodes exactes dépend très fortement de la taille des instances, c'est-à-dire du nombre de tâches. Résultats qui sont confirmés par le tableau 4.11 qui résume les résultats expérimentaux par nombre de tâches. En effet lorsqu'il n'y a que 12 tâches, la méthode directe résout seulement 40% des instances alors que les autres méthodes les résolvent toutes. Nous pouvons observer que la méthode inverse utilisant la programmation dynamique fournit très souvent de meilleures solutions que la méthode utilisant la PSE pour les deux étapes. Ainsi la méthode inverse utilisant la programmation dynamique résout 5 instances de 15 tâches sur 6 et presque 1/4 des instances de 20 tâches.

n	k	α	c_2	Heuristiques		Directe (PSE)						Inverse (dynamique)						Inverse (PSE)					
				Écart		Temps		Non	Non	Écart		Temps		Non	Non	Écart		Temps		Non	Non	Écart	
				Moy	Max	Moy	Max	Opt.	Best	Moy	Max	Moy	Max	Opt.	Best	Moy	Max	Moy	Max	Opt.	Best	Moy	Max
12	2	0.1	1	3,9	10,1	292	535	1	0	1,9	1,9	15	96	0	0	0,0	0,0	93	274	2	2	1,1	1,1
			2	4,9	8,3	75	178	7	6	6,6	13,6	2	6	0	0	0,0	0,0	3	8	0	0	0,0	0,0
			3	3,7	9,8	238	238	9	8	3,7	7,7	1	3	0	0	0,0	0,0	1	3	0	0	0,0	0,0
			4	5,5	12,1			10	8	3,7	9,9	1	3	0	0	0,0	0,0	1	2	0	0	0,0	0,0
		1	1	4,7	9,6	332	551	7	1	2,4	6,4	8	26	0	0	0,0	0,0	174	600	2	1	2,5	3,4
			2	7,0	15,3	515	515	9	7	11,9	15,8	1	5	0	0	0,0	0,0	6	23	0	0	0,0	0,0
			3	7,1	14,6			10	10	10,3	15,0	2	3	0	0	0,0	0,0	5	11	0	0	0,0	0,0
			4	4,1	8,2			10	10	8,9	11,9	1	2	0	0	0,0	0,0	1	3	0	0	0,0	0,0
	3	0.1	1	3,2	6,5	306	532	2	0	1,0	1,6	1	5	0	0	0,0	0,0	44	299	0	0	0,0	0,0
			2	5,4	12,7	176	416	4	3	4,8	8,2	0	0	0	0	0,0	0,0	0	1	0	0	0,0	0,0
			3	5,2	10,2	262	400	7	6	5,0	10,6	1	3	0	0	0,0	0,0	1	3	0	0	0,0	0,0
			4	4,5	12,6			10	10	5,4	9,1	1	3	0	0	0,0	0,0	1	3	0	0	0,0	0,0
		1	1	6,1	13,7	416	565	7	0	1,6	4,5	3	13	0	0	0,0	0,0	65	168	0	0	0,0	0,0
			2	6,1	10,8	195	447	7	1	10,2	5,4	0	1	0	0	0,0	0,0	9	23	0	0	0,0	0,0
			3	5,3	18,5			10	10	20,9	27,4	4	15	0	0	0,0	0,0	95	461	1	0	13,4	13,4
			4	4,5	12,7			10	7	19,7	26,5	0	1	0	0	0,0	0,0	14	52	0	0	0,0	0,0
20	2	0.1	1	2,9	6,7			9	**2**	0,8	2,4			7	5	1,1	2,6			9	7	1,1	2,6
			2	1,6	3,0			10	8	2,8	9,0	255	400	8	**0**	1,8	6,6	255	399	8	3	2,2	7,0
			3	3,0	7,2			10	9	2,9	10,7			10	**0**	1,7	6,2			10	1	1,8	6,4
			4	2,1	4,3			10	9	2,1	3,7			10	**0**	1,1	2,2			10	1	1,1	2,2
		1	1	2,2	4,8			10	**1**	0,7	1,8	0	0	8	4	1,2	2,2	1	1	8	5	1,2	2,2
			2	3,0	6,5			10	7	4,9	10,1	5	5	9	**0**	3,1	5,2			10	7	3,7	7,0
			3	3,2	7,6			10	8	4,9	8,5			10	**0**	3,1	3,8			10	3	3,4	5,0
			4	3,5	7,0			10	10	4,9	7,1			10	**0**	2,6	3,9			10	2	2,8	5,5
	3	0.1	1	1,7	3,8			10	**1**	0,5	1,8			6	2	0,8	1,8			6	2	0,8	1,8
			2	3,2	6,7			10	8	3,9	6,5	32	127	**2**	**0**	2,5	4,2			7	6	4,0	6,5
			3	3,9	8,0			10	10	6,0	15,5	60	250	**3**	**0**	1,9	2,4			5	3	4,5	10,5
			4	4,3	8,9			10	10	3,4	5,7	135	375	**3**	**0**	1,2	1,5			6	3	2,4	4,8
		1	1	3,1	7,1			8	**0**	0,4	0,8	1	3	5	4	1,6	2,7	15	531	5	4	1,6	2,7
			2	1,7	2,9			10	6	1,8	4,5	160	288	6	**2**	2,2	4,3			10	10	2,5	4,5
			3	7,9	11,3			10	10	13,8	19,4			10	**0**	9,1	12,3			10	10	13,2	18,4
			4	4,9	9,8			10	10	11,9	13,8			10	**0**	8,4	10,4			10	8	9,7	11,7

TABLE 4.7 – Comparaison des PSE lorsqu'il y a 2 machines sur chaque étage.

n	k	α	c_2	Heuristiques		Directe (PSE)						Inverse (dynamique)						Inverse (PSE)					
				Écart		Temps		Non	Non	Écart		Temps		Non	Non	Écart		Temps		Non	Non	Écart	
				Moy	Max	Moy	Max	Opt.	Best	Moy	Max	Moy	Max	Opt.	Best	Moy	Max	Moy	Max	Opt.	Best	Moy	Max
12	2	0,1	1	5,1	10,7	422	583	4	0	0,6	1,5	12	57	0	0	0,0	0,0	16	74	3	1	1,8	3,4
			2	4,7	9,9	98	281	1	1	9,7	9,7	5	43	0	0	0,0	0,0	60	509	0	0	0,0	0,0
			3	5,0	10,1	75	140	5	5	8,5	12,9	1	4	0	0	0,0	0,0	13	52	0	0	0,0	0,0
			4	7,8	15,4			10	7	15,4	22,3	0	0	0	0	0,0	0,0	2	11	0	0	0,0	0,0
		1	1	5,5	11,3	397	501	6	0	0,7	1,5	6	42	0	0	0,0	0,0	45	306	0	0	0,0	0,0
			2	5,7	10,5	91	214	0	0	0,0	0,0	5	13	0	0	0,0	0,0	102	299	0	0	0,0	0,0
			3	6,2	12,0	94	291	6	3	7,7	14,3	1	5	0	0	0,0	0,0	73	208	0	0	0,0	0,0
			4	6,3	13,7	413	413	9	6	12,1	28,9	1	1	0	0	0,0	0,0	30	61	0	0	0,0	0,0
	3	0,1	1	4,8	10,3	337	509	3	0	1,3	2,0	4	23	0	0	0,0	0,0	106	599	1	0	1,8	1,8
			2	3,7	9,1	69	238	0	0	0,0	0,0	2	13	0	0	0,0	0,0	25	174	0	0	0,0	0,0
			3	5,5	8,4	37	113	2	2	11,8	16,6	0	0	0	0	0,0	0,0	7	36	0	0	0,0	0,0
			4	7,4	19,0	0	0	9	7	14,1	23,0	0	0	0	0	0,0	0,0	1	4	0	0	0,0	0,0
		1	1	6,0	9,7	307	327	7	0	0,8	2,8	2	6	0	0	0,0	0,0	78	593	0	0	0,0	0,0
			2	8,3	19,7	75	251	0	0	0,0	0,0	0	1	0	0	0,0	0,0	58	140	0	0	0,0	0,0
			3	7,8	13,4	80	139	6	2	8,5	21,6	1	3	0	0	0,0	0,0	185	589	0	0	0,0	0,0
			4	6,5	12,6	249	447	6	4	11,6	22,0	0	0	0	0	0,0	0,0	42	135	0	0	0,0	0,0
20	2	0,1	1	2,7	4,3			9	**0**	0,4	0,7	1	2	**5**	3	0,9	1,7	1	2	**5**	3	0,9	1,7
			2	2,4	4,3			10	**0**	0,7	3,0			10	2	0,9	3,0			10	2	0,9	3,0
			3	1,8	3,3			10	2	1,5	3,2	1	2	8	**2**	1,6	2,6			10	8	2,2	3,4
			4	5,9	11,3			10	10	9,4	21,8	164	495	**4**	**0**	4,2	5,9	1	1	9	9	10,6	21,6
		1	1	3,4	7,1			10	**0**	0,4	0,6	233	274	8	7	1,0	1,5	233	274	8	7	1,0	1,5
			2	2,9	5,2			10	**0**	0,4	0,9	224	231	8	3	1,1	2,9	224	231	8	3	1,1	2,9
			3	3,0	5,5			10	**0**	3,9	12,6			10	9	5,5	12,6			10	9	5,6	12,6
			4	3,5	6,7			10	**3**	7,7	21,8			10	3	7,6	21,8			10	3	8,8	23,0
	3	0,1	1	2,2	6,8			9	**0**	0,3	0,4	0	0	**4**	3	0,6	0,8	0	0	**4**	3	0,6	0,8
			2	2,6	4,8			10	**1**	1,2	3,6			10	6	1,8	3,8			10	7	2,1	5,3
			3	2,6	6,2			10	**2**	1,2	3,4			7	3	1,4	2,7			10	8	1,8	3,8
			4	3,1	6,7			10	7	6,6	15,7			**5**	**1**	5,8	14,9			10	10	7,5	15,7
		1	1	2,5	4,7	0	0	8	**0**	0,4	0,6	0	0	8	6	1,6	4,5	0	0	8	6	1,6	4,5
			2	3,5	6,2			10	**0**	0,8	1,9	363	455	8	7	0,4	0,8	363	455	8	7	0,4	0,8
			3	3,4	7,2			10	**0**	0,7	2,4			9	8	3,3	7,0			9	9	3,5	7,0
			4	3,2	5,0			10	**0**	1,7	4,3			10	10	4,2	8,1			10	10	4,2	8,1

TABLE 4.8 – Comparaison des PSE lorsqu'il y a 2 machines sur le premier étage et 5 sur le second.

n	k	α	c_2	Heuristiques		Directe (PSE)						Inverse (dynamique)						Inverse (PSE)					
				Écart		Temps		Non	Non	Écart		Temps		Non	Non	Écart		Temps		Non	Non	Écart	
				Moy	Max	Moy	Max	Opt.	Best	Moy	Max	Moy	Max	Opt.	Best	Moy	Max	Moy	Max	Opt.	Best	Moy	Max
12	2	0,1	1	4,6	11,9	291	527	2	0	3,5	6,7	2	12	0	0	0,0	0,0	2	12	0	0	0,0	0,0
			2	7,4	12,4			10	2	3,2	10,1	1	4	0	0	0,0	0,0	1	4	0	0	0,0	0,0
			3	6,2	12,4			10	0	1,9	6,4	4	24	0	0	0,0	0,0	4	24	0	0	0,0	0,0
			4	4,2	7,3	0	0	9	1	2,0	5,8	1	3	0	0	0,0	0,0	1	3	0	0	0,0	0,0
		1	1	8,4	13,4	101	131	7	5	10,1	12,9	2	6	0	0	0,0	0,0	2	6	0	0	0,0	0,0
			2	7,8	12,2			10	6	7,3	10,0	3	6	0	0	0,0	0,0	3	6	0	0	0,0	0,0
			3	5,1	10,9			10	6	5,2	7,3	2	3	0	0	0,0	0,0	2	3	0	0	0,0	0,0
			4	3,7	6,1			10	1	4,2	7,5	2	5	0	0	0,0	0,0	2	5	0	0	0,0	0,0
	3	0,1	1	6,1	11,3	171	472	0	0	0,0	0,0	0	1	0	0	0,0	0,0	0	1	0	0	0,0	0,0
			2	5,7	11,2	329	597	6	1	2,2	5,0	1	2	0	0	0,0	0,0	1	2	0	0	0,0	0,0
			3	6,1	12,0			10	2	2,7	5,9	1	3	0	0	0,0	0,0	1	3	0	0	0,0	0,0
			4	4,8	12,5			10	5	3,5	5,7	1	1	0	0	0,0	0,0	1	1	0	0	0,0	0,0
		1	1	7,1	12,8	181	537	4	2	22,2	25,9	0	0	0	0	0,0	0,0	1	2	0	0	0,0	0,0
			2	8,3	13,9			10	8	20,5	22,8	1	2	0	0	0,0	0,0	1	1	0	0	0,0	0,0
			3	4,9	9,6			10	9	12,1	18,7	1	1	0	0	0,0	0,0	1	1	0	0	0,0	0,0
			4	4,5	10,0			10	7	13,6	20,0	0	1	0	0	0,0	0,0	0	1	0	0	0,0	0,0
20	2	0,1	1	3,9	9,5			9	7	5,6	14,7			9	5	4,2	11,6			9	**0**	2,5	4,8
			2	3,8	7,5			10	5	2,3	4,6			10	**1**	1,6	3,3			10	**1**	1,6	3,3
			3	4,8	7,4			10	9	2,8	5,0			10	**1**	1,3	2,6			10	**1**	1,3	2,6
			4	3,1	7,0			10	8	1,7	2,8			10	**0**	1,0	1,6			10	**0**	1,0	1,6
		1	1	3,8	11,9			10	6	7,2	10,0			10	4	6,9	9,0			10	**1**	6,1	7,6
			2	5,2	6,8			10	10	4,9	8,0			10	**0**	2,8	5,5			10	**0**	2,8	5,5
			3	3,1	4,8			10	9	4,3	5,9			10	**0**	2,5	4,9			10	**0**	2,5	4,9
			4	3,1	4,5			10	10	2,9	4,2			10	**0**	1,8	2,7			10	**0**	1,8	2,7
	3	0,1	1	6,5	16,5			10	9	7,5	13,1	291	578	**4**	**0**	3,4	7,6	319	559	4	1	4,5	7,6
			2	5,1	10,5			10	10	6,8	11,2	246	343	3	0	2,2	2,8	203	351	**2**	**0**	1,8	2,6
			3	5,1	9,5			10	9	3,8	7,1	207	460	**4**	**0**	1,2	2,0	168	449	**4**	**0**	1,2	2,0
			4	3,8	7,7			10	9	3,1	7,2	248	480	4	0	0,9	1,5	267	597	**3**	**0**	0,9	1,5
		1	1	9,5	14,7			10	9	12,9	20,9			9	**3**	12,4	21,9			10	5	10,9	22,8
			2	7,5	15,3			10	9	13,2	16,1			10	4	9,9	16,8			10	**1**	9,1	13,2
			3	8,1	12,7			10	10	7,9	10,4			10	**0**	5,2	7,7			10	**0**	5,2	7,7
			4	7,6	12,0			10	10	7,9	9,9			10	**0**	5,4	8,1			10	**0**	5,4	8,1

TABLE 4.9 – Comparaison des PSE lorsqu'il y a 5 machines sur le premier étage et 2 sur le second.

n	k	α	c_2	Heuristiques		Directe (PSE)						Inverse (dynamique)						Inverse (PSE)					
				Écart		Temps		Non	Non	Écart		Temps		Non	Non	Écart		Temps		Non	Non	Écart	
				Moy	Max	Moy	Max	Opt.	Best	Moy	Max	Moy	Max	Opt.	Best	Moy	Max	Moy	Max	Opt.	Best	Moy	Max
12	2	0,1	1	12,4	19,9	17	90	0	0	0,0	0,0	1	5	0	0	0,0	0,0	5	28	0	0	0,0	0,0
			2	8,3	21,5	0	0	8	8	14,3	23,8	0	0	0	0	0,0	0,0	0	0	0	0	0,0	0,0
			3	7,3	19,6	0	0	9	8	8,3	18,5	0	0	0	0	0,0	0,0	0	0	0	0	0,0	0,0
			4	6,2	15,2	94	376	6	5	11,6	27,7	0	0	0	0	0,0	0,0	0	0	0	0	0,0	0,0
		1	1	11,4	22,1	17	131	1	0	10,3	10,3	4	15	0	0	0,0	0,0	76	277	0	0	0,0	0,0
			2	3,2	9,9	34	202	4	3	2,4	5,4	1	5	0	0	0,0	0,0	6	44	0	0	0,0	0,0
			3	7,7	21,0	16	63	6	6	6,9	14,4	0	0	0	0	0,0	0,0	1	3	0	0	0,0	0,0
			4	5,2	15,3	2	7	3	3	9,2	15,6	0	2	0	0	0,0	0,0	0	1	0	0	0,0	0,0
	3	0,1	1	12,5	22,9	1	4	0	0	0,0	0,0	2	8	0	0	0,0	0,0	30	200	0	0	0,0	0,0
			2	2,7	5,2	89	375	4	3	9,1	16,5	0	1	0	0	0,0	0,0	0	0	0	0	0,0	0,0
			3	5,8	27,0	45	272	4	3	7,0	14,1	0	0	0	0	0,0	0,0	0	0	0	0	0,0	0,0
			4	4,1	13,4	6	24	6	4	3,5	7,4	0	1	0	0	0,0	0,0	0	0	0	0	0,0	0,0
		1	1	18,1	27,6	14	72	0	0	0,0	0,0	1	4	0	0	0,0	0,0	18	67	0	0	0,0	0,0
			2	4,3	9,4	45	426	0	0	0,0	0,0	0	1	0	0	0,0	0,0	2	16	0	0	0,0	0,0
			3	3,4	8,2	1	2	2	2	2,1	3,1	1	3	0	0	0,0	0,0	2	11	0	0	0,0	0,0
			4	2,1	11,1	0	1	1	1	1,3	1,3	0	1	0	0	0,0	0,0	0	3	0	0	0,0	0,0
20	2	0,1	1	8,4	15,2			10	**0**	5,8	14,8			10	8	10,4	19,0			10	8	8,9	15,8
			2	10,0	22,7			10	10	19,1	28,6	294	469	6	3	14,1	27,8	170	254	6	**2**	13,8	20,6
			3	12,6	20,2			10	10	12,4	19,7	386	589	7	3	6,9	10,4	300	401	6	**0**	7,1	10,1
			4	11,4	16,3			10	10	11,5	22,3	160	515	2	2	8,8	3,4	184	526	**1**	**0**	3,8	3,8
		1	1	12,3	21,0			10	**1**	4,4	12,3			10	8	5,2	11,8			10	8	5,3	11,8
			2	8,3	12,7			10	9	22,4	30,7			10	**2**	18,8	24,9			10	5	18,5	24,3
			3	8,8	14,7			10	10	20,4	25,1			10	2	14,5	25,4			10	**1**	13,4	15,3
			4	9,2	13,7			10	10	19,0	24,4			10	**0**	11,7	16,4			10	1	11,7	16,4
	3	0,1	1	10,5	18,5			10	**0**	4,3	7,6			10	9	6,3	13,3			10	9	6,5	13,3
			2	10,0	13,3			10	8	18,4	26,1	103	509	**3**	**3**	21,8	28,6	60	70	6	5	15,4	28,6
			3	14,9	22,0			10	10	18,4	26,6	60	135	**2**	**1**	13,1	20,9	79	200	**2**	**1**	15,6	16,2
			4	13,5	32,3			10	10	18,1	21,8	31	114	**0**	**0**	0,0	0,0	5	15	**0**	**0**	0,0	0,0
		1	1	9,0	15,0			10	**0**	4,0	8,6			10	**0**	4,0	8,6			10	10	6,5	15,5
			2	6,0	13,6			10	8	21,1	31,2			10	**2**	18,3	29,9			10	6	19,4	29,9
			3	7,7	16,6			10	6	14,7	34,3			10	**0**	11,3	22,5			10	5	12,9	27,9
			4	8,6	17,8			9	7	14,2	22,7			9	**1**	11,0	17,3			10	7	11,4	17,5

TABLE 4.10 – Comparaison des PSE lorsqu'il y a 5 machines sur le premier étage et 5 sur le second.

n	Heuristiques		Directe (PSE)						Inverse (dynamique)						Inverse (PSE)					
	Écart		Temps		%		Écart		Temps		%		Écart		Temps		%		Écart	
	Moy	Max	Moy	Max	Opt.	Best	Moy	Max	Moy	Max	Opt.	Best	Moy	Max	Moy	Max	Opt.	Best	Moy	Max
12	6,0	27,6	167	597	41	65	6,6	28,9	2	96	100	100	0,0	0,0	24	600	99	99	0,3	13,4
15	6,6	23,3	23	563	10	39	7,1	35,1	63	577	84	95	1,6	12,4	90	599	64	81	3,1	23,0
20	5,4	32,3	0	0	1	40	6,7	34,3	156	589	23	76	5,1	29,9	145	597	18	60	5,3	29,9

TABLE 4.11 – Comparaison des PSE en fonction de n.

4.5 Conclusion du chapitre

Ce chapitre a été l'occasion de présenter deux méthodes exactes de type procédure par séparation évaluation pour le problème de minimisation de la durée totale de l'ordonnancement d'un flowshop hybride avec machines à traitement par batches et compatibilité entre les tâches, problème noté $HF2(m_1, Bm_2)|G_p = INT, b = k|C_{max}$. Les résultats expérimentaux, présentés dans la section 4.4, indiquent que la meilleure méthode résout 1/4 des instances lorsqu'il y a 20 tâches et 5/6 des instances composées de 15 tâches en moins de 600 secondes.

Nous avons présenté la méthode directe lors des conférences ROADEF'08 et PMS'08 et les deux types de méthodes lors des conférences ROADEF'09 et EURO XXIII.

Chapitre 5

Autres critères réguliers

Dans ce chapitre nous nous intéressons aux problèmes de minimisation de critères réguliers pour une machine à traitement par batches, compatibilité entre les tâches et capacité de la machine infinie ($B|G_p^\alpha = INT, b \geqslant n|\gamma$). Pour rappel, un critère est dit *régulier* si la réduction de la date de fin d'exécution d'une tâche, sans modification de l'ordonnancement des autres tâches, implique une réduction de la valeur de la fonction objectif. Les critères réguliers classiques sont les critères que nous avons introduits lors de la présentation du champs γ de la notation des problèmes d'ordonnancement de la production $\alpha|\beta|\gamma$. C'est-à-dire le makespan C_{max}, le retard algébrique maximum L_{max}, le plus grand retard T_{max}, et les sommes pondérées ou non des dates de fin de traitement $\sum C_j$ et $\sum w_j C_j$, des retards $\sum T_j$ et $\sum w_j T_j$ et du nombre de tâches en retard $\sum U_j$ et $\sum w_j U_j$. Nous supposons que les durées maximales d'exécution sont proportionnelles aux durées minimales ce qui est le cas du problème industriel (voir 1.4) qui nous a inspiré.

La première section de ce chapitre présente des propriétés communes de certains ordonnancements optimaux de tout problème $B|G_p^\alpha = INT, b \leqslant n|\gamma$. Nous abordons rapidement le problème de minimisation de la date de fin d'ordonnancement, puis nous développons un algorithme de programmation dynamique pour le problème de minimisation de la somme des dates de fin d'exécution. Finalement après avoir démontré la NP-difficulté des problèmes de minimisation de critères réguliers basés sur des dates de fin souhaitées nous présentons deux algorithmes de programmation dynamique similaires pour la minimisation du plus grand retard algébrique, et la minimisation de la somme pondérée des retards.

Sommaire

5.1 Propriétés des ordonnancements optimaux 106
5.2 Minimisation du Makespan 107
5.3 Minimisation de la somme des dates de fin d'exécution 107
5.3.1 Nombre de batches reportés 108
5.3.2 Algorithme de programmation dynamique 110
5.4 Minimiser des fonctions objectif avec dates de fin souhaitées 112
5.4.1 Algorithme de programmation dynamique 115
5.5 Conclusion du chapitre 117

Nous considérons le problème de minimisation de critères réguliers pour une machine à traitement par batches avec intervalles proportionnels de durées d'exécution compatibles et capacité de la machine infinie. Pour rappel, comme les intervalles de durées d'exécution sont proportionnels et doivent être compatibles, la durée d'exécution du batch contenant la tâche j doit appartenir à l'intervalle de durées d'exécution de la tâche j, $[a_j, (1+\alpha)a_j]$.

Brucker et al. [28] ont étudié la complexité des problèmes de minimisation de critères réguliers pour une machine à traitement par batches sans contraintes de compatibilité $B|b \geqslant n|\gamma$ et $B|b = k|\gamma$, dans ce chapitre nous effectuons ce même type d'étude lorsque les intervalles de durées d'exécution sont proportionnels et compatibles. Finke et al. [74] ont étudié la minimisation du makespan pour plusieurs types de graphes de compatibilités et plusieurs types de capacité et de durées d'exécution des batches. Ils ont notamment proposé un algorithme de programmation dynamique de complexité $O(n^3)$ pour le problème $B|G = INT|C_{max}$.

5.1 Propriétés des ordonnancements optimaux

Dans cette section nous présentons des propriétés applicables à tous les critères réguliers. Afin de simplifier les notations nous supposons pour l'ensemble de cette section que les tâches sont indexées suivant la règle SPT (Shortest-Processing-Time), c'est-à-dire par durée d'exécution croissante ($a_1 \leqslant \cdots \leqslant a_n$).

Lemme 5.1.1 *Pour toute fonction objectif régulière à minimiser, il existe un ordonnancement optimal $(B_1, \ldots, B_r)$ tel que chaque batch est composé de tâches d'indices consécutifs, c'est-à-dire $B_k = \{i_k, i_k + 1, \ldots, i_k + |B_k| - 1\}$, $k = 1, \ldots, r$.*

Preuve. Soit σ un ordonnancement optimal tel que le lemme 5.1.1 n'est pas vérifié, c'est-à-dire qu'il existe deux batches B et G, et une tâche $j \in G$ tel que $\min\{i | i \in B\} < j < \max\{i | i \in B\}$.

i. Si B précède G, alors le déplacement de la tâche j du batch G vers le batch B est possible car la contrainte de compatibilité est respectée. De plus cette opération permet de réduire la date de fin d'exécution de la tâche j sans augmenter les dates de fin d'exécution des autres tâches.

ii. Si G précède B, alors le déplacement des tâches appartenant à B d'indice inférieur à j ($\forall i \in B, i < j$) vers le batch G est possible car les contraintes de compatibilité sont respectées. De plus cette opération permet de réduire les dates de fin d'exécution des tâches déplacées sans modifier l'ordonnancement des autres tâches.

L'ordonnancement obtenu par cette transformation est au moins aussi bon que l'ordonnancement d'origine, c'est-à-dire optimal. Ainsi, une succession d'un nombre fini de transformations de ce type permet d'obtenir un ordonnancement optimal respectant le lemme 5.1.1.

□

Lemme 5.1.2 *Pour toute fonction objectif régulière à minimiser, il existe un ordonnancement optimal dans lequel les tâches de même durée d'exécution sont affectées au même batch.*

Preuve. Si des tâches de même durée d'exécution sont ordonnancées dans plusieurs batches, alors elles peuvent toutes être déplacées vers le batch exécuté en premier sans augmentation de la valeur de la fonction objectif.

□

Lemme 5.1.3 *Pour toute fonction objectif régulière à minimiser, il existe un ordonnancement optimal tel que le batch $B = \{i, i+1, \ldots, j\}$ est séquencé après le batch $G = \{j+1, j+2, \ldots, k\}$ si et seulement si j et k ne sont pas compatibles, c'est-à-dire $(1+\alpha)a_j < a_k$.*

Preuve. Soit σ un ordonnancement optimal ne respectant pas le lemme 5.1.3, alors le batch $B = \{i, i+1, \ldots, j\}$ est séquencé après le batch $G = \{j+1, j+2, \ldots, k\}$ et $(1+\alpha)a_j \geqslant a_k$. Le déplacement de la tâche j dans le batch G est possible, car la contrainte de compatibilité est respectée, de plus cela réduit la date de fin d'exécution de la tâche j sans modifier l'ordonnancement des autres tâches. Un nombre fini de répétition de cette transformation permet d'obtenir un ordonnancement optimal respectant le lemme.
□

Définition Un batch $B = \{i, i+1, \ldots, j\}$ est dit *g-complet* (complet du côté gauche) si $(1+\alpha)a_{i-1} < a_j$, c'est-à-dire s'il contient toutes les tâches compatibles avec sa durée d'exécution. Un batch B est dit *reporté par rapport* au batch G si le batch G précède le batch B et $p(B) < p(G)$, le batch G est quand à lui appelé batch *reportant*. D'après le lemme 5.1.3, pour toute fonction objectif régulière, un batch reporté est reporté par rapport à, au moins, un batch g-complet.

5.2 Minimisation du Makespan

Ce problème a déjà été traité par Oulamara et al. [176], c'est pourquoi nous abordons rapidement les résultats concernant la minimisation du makespan. Soit σ un ordonnancement quelconque. Pour la minimisation du makespan, l'ordre des batches étant sans importance. Supposons que le premier batch contient la tâche d'indice n (de plus grande durée d'exécution). Toute tâche compatible avec la tâche d'indice n peut être déplacée dans le premier batch sans augmenter la valeur de C_{max}, mais aucune tâche incompatible avec la tâche d'indice n ne peut être ajoutée à ce batch. C'est pourquoi il existe un ordonnancement optimal dans lequel le premier batch contient la tâche n et toutes les tâches compatibles avec la tâche d'indice n, ainsi la tâche d'indice n permet à elle seule de définir le premier batch. Le premier batch créé, nous supprimons toutes ses tâches de l'ensemble de tâche puis nous créons le premier batch du nouvel ensemble de tâches de la même manière que pour le batch précédent. Ce nouveau batch est le second batch d'un ordonnancement optimal du problème d'origine. Ainsi si les tâches sont déjà triées par ordre décroissant de leur durée minimale d'exécution (règle SPT), la création d'un ordonnancement optimal nécessite $O(n)$ unités de temps. Le problème $B|G_p = INT, b \geqslant n|C_{max}$ est donc résolu en $O(n \log n)$ unités de temps.

5.3 Minimisation de la somme des dates de fin d'exécution

Dans cette section nous présentons une méthode utilisant la programmation dynamique pour résoudre le problème de minimisation de la somme des dates de fin d'exécution des tâches $B|G_p^{\alpha} = INT, b \geqslant n|\sum C_j$. Cette méthode est une extension de la méthode présentée par Brucker et al. [28] pour le problème $B|b = k|\sum C_j$.

Notons que lorsque les batches sont donnés, les contraintes de compatibilité n'ont aucune influence sur le séquencement optimal de ces batches. Dans ce cas l'ordre des batches est donnés par le lemme 5.3.1, de Chandru et al. [33], pour le problème sans contraintes de compatibilité $(B||\sum C_j)$.

Lemme 5.3.1 *Pour un ensemble de batches $B_1, B_2, \ldots, B_r$, une séquence $(B_1, B_2, \ldots, B_r)$ est optimale pour le critère $\sum C_j$ si et seulement si :*

$$\frac{p(B_1)}{|B_1|} \leq \frac{p(B_2)}{|B_2|} \leq \ldots \leq \frac{p(B_r)}{|B_r|}.$$

Preuve. Supposons que la date de fin d'exécution du batch B_i est égale à C_i. Comme toutes les tâches du batch sont terminées à cette date, la contribution du batch dans $\sum w_j C_j$ est égale à $|B_i|C_i$. C'est pourquoi le batch B_i peut être considérer lors du séquencement comme une tâche de durée $p(B_i)$ et de poids $|B_i|$, nous obtenons ainsi un problème de minimisation de la somme pondérée des dates de fin d'exécution $1||\sum w_j C_j$ où les r batches correspondent aux r tâches du nouveau problème. Ce problème est résolu optimalement à l'aide de la règle WSPT (Weighted-Shortest-Processing-Time), Smith [194] .

□

Il est à noter que cette règle provoque l'apparition de batches reportés dans la séquence optimale, comme illustré dans l'exemple ci-dessous.

5.3.1 Nombre de batches reportés

S'il n'y avait pas de batches reportés par rapport à d'autres batches dans un ordonnancement optimal, alors les batches seraient ordonnancés suivant leur durée d'exécution par ordre croissant, c'est-à-dire suivant l'ordre *SPT-batch*. Dans ce cas un algorithme de programmation dynamique peut être facilement développable pour le problème $B|G_p^\alpha = INT, b \geqslant n|\sum C_j$. Nous avons démontré précédemment qu'un batch reporté ne peut pas être reporté uniquement par rapport à des batches non g-complets. C'est-à-dire qu'ils sont reportés par rapport à un batch g-complet. Dans cette section nous allons borner le nombre de batch reporté par rapport à un batch B g-complet.

Soient B un batch g-complet, et D un batch reporté par rapport au batch B. Par définition, B précède D et $p(D) < p(B)$. D'après le lemme 5.3.1, l'assertion B précède D implique que $\frac{p(B)}{|B|} \leqslant \frac{p(D)}{|D|}$ et $|B| > |D|$. De plus $|D|$ contient au moins une tâche $\frac{p(B)}{|B|} \leqslant p(D) < p(B)$ donc la durée d'exécution de tout batch reporté par rapport au batch B appartient à l'intervalle $[\frac{p(B)}{|B|}, p(B))$. Nous développons ici une borne supérieure du nombre de batches D reportés par rapport à un batch g-complet B, les durées d'exécution de ces batches appartiennent à l'intervalle $[\frac{p(B)}{|B|}, p(B))$.

Partitionnons les batches reportés par rapport à un batch g-complet B en ensembles $S_1, \ldots, S_t$ tel qu'un batch $D \in S_i$ est reporté par rapport au batch B si $\frac{p(B)}{|B|}(1 + \min\{1, \alpha\})^{i-1} \leqslant p(D) \leqslant \frac{p(B)}{|B|}(1 + \min\{1, \alpha\})^i$, $i = 1, \ldots, t$, où $t = \min\{j \mid \frac{p(B)}{|B|}(1 + \min\{1, \alpha\})^j \geqslant p(B)\}$. C'est-à-dire $t = \lceil \frac{\log_2 |B|}{\log_2(1+\min\{1,\alpha\})} \rceil$.

Le lemme 5.3.2 permet d'évaluer le nombre de batches reportés dans chaque ensemble S_i, $i = 1, \ldots, t$, et ainsi de borner le nombre de batches reportés par rapport à un batch B dans le lemme 5.3.3.

Lemme 5.3.2 *Soit $S_j = \{D_1, \ldots, D_v\}$, supposons les batches D_i indexés suivant l'ordre SPT. Il existe un ordonnancement optimal du problème $B|G_p^\alpha = INT|\sum C_j$ dans lequel l'inégalité $|d_{i+1}| < \min\{1, \alpha\}|D_i|, i = 2, \ldots, v-1$ est vérifiée.*

Preuve. Remarquons tout d'abord que les plus grandes tâches des batches $D_1, \ldots, D_v$ sont compatibles. De plus, d'après le lemme 5.1.3 les batches $D_1, \ldots, D_v$ apparaissent dans cet ordre dans un ordonnancement optimal σ. Soit $2 \leqslant i \leqslant v-1$.

Soit $\sigma = (X_1, D_i, X_2, D_{i+1}, X_3)$ un ordonnancement optimal tel que le nombre de batches reportés par rapport au batch B soit minimal, où X_1 est un bloc de batches contenant les batches B et $D_1, \ldots, D_{i-1}$, et X_3 est un bloc de batches contenant $D_{i+2}, \ldots, D_v$. Notons $p(X_j)$ et $|X_j|$ la durée d'exécution totale et le nombre de tâches du bloc X_j, pour $j = 1, 2, 3$. Considérons un nouvel ordonnancement σ' obtenu à partir de σ en regroupant les batches D_i et D_{i+1} au sein du batch D' contenant $|D_i| + |D_{i+1}|$ tâches et de durée d'exécution $p(D') = p(D_{i+1})$. Les batches de σ' sont séquencés comme suit $\sigma' = (X_1, D', X_2, X_3)$, notons que cet ordonnancement peut ne pas respecter le lemme 5.3.1, il peut donc être amélioré par la suite.

Notons que D_1 et D_2 ne peuvent être groupés puisque D_1 peut contenir des tâches incompatibles avec les tâches de D_2, d'où le choix de $2 \leqslant i \leqslant v-1$.

Calculons

$$\sum C_j(\sigma) - \sum C_j(\sigma') = |D_{i+1}|p(X_2) - |X_2|(p(D_{i+1}) - p(D_i))$$
$$+(|D_{i+1}| + |X_3|)p(D_i) - |D_i|(p(D_{i+1}) - p(D_i)).$$

Supposons que $|D_{i+1}| \geqslant \min\{1, \alpha\}|D_i|$ et montrons que σ' est au moins aussi bon que σ, ce qui contredit l'optimalité de σ.

Comme les plus longues tâches de D_i et D_{i+1} sont compatibles et appartiennent au même ensemble S_i, nous avons $p(D_{i+1}) - p(D_i) \leqslant \min\{1, \alpha\}p(D_i)$. Il est alors nécessaire de distinguer deux cas.

1. $\min\{1, \alpha\}|X_2|p(D_i) \leqslant |D_{i+1}|p(X_2)$. Or $p(D_{i+1}) - p(D_i) \leqslant \min\{1, \alpha\}p(D_i)$ d'où $|X_2|(p(D_{i+1}) - p(D_i)) \leqslant |X_2| \min\{1, \alpha\}p(D_i) \leqslant |D_{i+1}|p(X_2)$. Ainsi :

$$\begin{aligned}
\sum C_j(\sigma) - \sum C_j(\sigma') &= |D_{i+1}|p(X_2) - |X_2|(p(D_{i+1}) - p(D_i)) + (|D_{i+1}| + |X_3|)p(D_i) \\
&\quad -|D_i|(p(D_{i+1}) - p(D_i)) \\
&\geqslant |D_{i+1}|p(X_2) - |X_2| \min\{1, \alpha\}p(D_i) + (|D_{i+1}| + |X_3|)p(D_i) \\
&\quad -|D_i| \min\{1, \alpha\}p(D_i) \\
&\geqslant |D_{i+1}|p(X_2) - |D_{i+1}|p(X_2) + (|D_{i+1}| + |X_3|)p(D_i) \\
&\quad -|D_i| \min\{1, \alpha\}p(D_i) \\
&\geqslant (|D_{i+1}| - |D_i| \min\{1, \alpha\})p(D_i) \geqslant 0.
\end{aligned}$$

   Ce qui contredit l'optimalité de σ.

2. $\min\{1, \alpha\}|X_2|p(D_i) > |D_{i+1}|p(X_2)$. Comme D_i est ordonnancé avant X_2 dans σ, $|X_2|p(D_i) \leqslant |D_i|p(X_2)$ et $\min\{1, \alpha\}|X_2|p(D_i) \leqslant \min\{1, \alpha\}|D_i|p(X_2)$. C'est pourquoi nous supposons que $\min\{1, \alpha\}|X_2|p(D_i) > |D_{i+1}|p(X_2)$ ce qui implique $\min\{1, \alpha\}|D_i|p(X_2) > |D_{i+1}|p(X_2)$ et $\min\{1, \alpha\}|D_i| > |D_{i+1}|$, qui contredit les suppositions initiales.

C'est pourquoi il existe un ordonnancement optimal avec un minimum de batches reportés par rapport au batch B tel que $|D_{i+1}| < \min\{1, \alpha\}|D_i|$, $i = 2, \ldots, v-1$.

□

Soit b^* la cardinalité du plus grand batch d'un ordonnancement optimal, b^* ne peut excéder la taille de la clique maximale du graphe d'intervalles G. Cette taille peut être obtenue en $O(n \log n)$, ou en $O(n)$ lorsque les intervalles sont triés ([94]). Le lemme précédent, accompagné

de cette nouvelle notation, nous permet de limiter le nombre de batches reportés par rapport à un même batch.

Lemme 5.3.3 *Quelque soit l'ordonnancement, le nombre de batches reportés par rapport à un batch g-complet est inférieur à :*

$$r^* = \begin{cases} \left\lceil \frac{\log_2 b^*}{\log_2(1+\alpha)} \right\rceil \frac{\log_2(b^*-1)}{\log_2 \frac{1}{\alpha}} & si\ \alpha < 1, \\ \lceil \log_2 b^* \rceil (b^*-1) & si\ \alpha \geqslant 1. \end{cases}$$

Preuve.

Soit σ un ordonnancement optimal, et B un batch reportant de σ. Ce batch contient au plus b^* tâches.

Quel que soit l'ensemble $S_i,\ i = 1, \ldots, t$ de batches reportés par rapport au batch B, notons $D_1, \ldots, D_{r_{S_i}}$ les batches appartenant à cet ensemble S_i. Comme les batches sont ordonnancés par ordre croissant du ratio $\frac{p(B_i)}{|B_i|}$, on a $|D_2| \leqslant |B| - 1$. Deux cas sont alors possibles :

- Si $\alpha < 1$ alors, d'après le lemme 5.3.2 $\alpha|D_i| > |D_{i+1}|,\ i = 2, \ldots, r_{S_i} - 1$. D'où $(|B| - 1)\alpha^{r_{S_i}-2} \geqslant |D_2|\alpha^{r_{S_i}-2} > |D_{r_{S_i}}| \geqslant 1$, ainsi $r_{S_i} < \frac{\log_2(|B|-1)}{\log_2 1/\alpha} + 2$ et $r_{S_i} \leqslant \frac{\log_2(|B|-1)}{\log_2 1/\alpha} + 1$.
- Si $\alpha \geqslant 1$ alors, d'après le lemme 5.3.2 $|D_i| > |D_{i+1}|,\ i = 2, \ldots, r-1$. Or $|D_{r_{S_i}}| \leqslant 1$, donc $r_{S_i} \leqslant |B|$.

Or le nombre d'ensemble de batches reportés $S_i,\ i = 1, \ldots, t$ par rapport au batch B est inférieur a $t = \lceil \frac{\log_2 |B|}{\log_2(1+\min\{1,\alpha\})} \rceil$, d'où :

$$r^* = \begin{cases} \left\lceil \frac{\log_2 b^*}{\log_2(1+\alpha)} \right\rceil \frac{\log_2(b^*-1)}{\log_2 \frac{1}{\alpha}} & \text{si } \alpha < 1, \\ \lceil \log_2 b^* \rceil (b^*-1) & \text{si } \alpha \geqslant 1. \end{cases}$$

□

Dans la section suivante nous décrivons un algorithme de programmation dynamique permettant de minimiser la somme des dates de fin d'exécution. Par la suite r^* désigne le nombre maximum de batches reportés par rapport à un même batch.

5.3.2 Algorithme de programmation dynamique

L'algorithme de programmation dynamique que nous présentons dans cette section est une adaptation de l'algorithme fourni par Brucker et al. [28] pour la minimisation de la somme des dates de fin d'exécution sur une machine à traitement par batches sans contraintes de compatibilité $B||\sum C_j$.

Soit r le nombre de batches reportés par rapport à un batch g-complet. La *tâche de tête* i_k du batch B_{i_k} désigne la tâche du batch B_{i_k} de plus grande durée d'exécution. Si le batch B_{i_k} est g-complet nous le notons $B^*_{i_k}$.

Soit σ un ordonnancement contenant les tâches $j, \ldots, n$ mais pas la tâche $j-1$, et contenant les batches $B_{i_1}, \ldots, B_{i_r}$ qui sont reportés par rapport au batch B_{j-1}. Pour rappel, la tâche de tête du batch B_{j-1} est $j-1$ et ce batch doit être ordonnancé plus tard. Les batches $B_{i_1}, \ldots, B_{i_r}$ sont indexés par ratio $\frac{p(B_i)}{|B_i|}$ croissant. Afin de respecter le lemme 5.3.1, ces batches doivent être

séquencés dans cet ordre dans l'ordonnancement σ, de plus le lemme 5.3.3 indique que le nombre de batches reportés est inférieur à r^*. Remarquons que si $r = 0$ alors aucun batch n'est reporté. Le lemme 5.1.1, nous permet de supposer que chaque batch B_{i_k} est constitué de tâches d'indices consécutifs, c'est-à-dire $B_{i_k} = \{i_{k'}, i_{k'+1}, \ldots, i_k\}$, $k = 1, \ldots, r$.

On dit que l'ordonnancement σ est dans l'état $(j, B_{i_1}, \ldots, B_{i_r})$, s'il contient les tâches d'indice supérieur à j et les batches $B_{i_1}, \ldots, B_{i_r}$ sont reportés par rapport au batch B_{j-1}. Si $r = 0$, on dit que σ est dans l'état $(j, \emptyset)$, où $\emptyset$ indique qu'aucun batch n'est reporté par rapport au batch B_{j-1} dans l'ordonnancement σ.

Un batch B peut être ajouté à un ordonnancement partiel σ si toutes les tâches de B sont compatibles. Afin d'obtenir un algorithme de programmation dynamique les batches sont ajoutés au début de l'ordonnancement précédent σ. De plus le lemme 5.3.1 impose qu'un batch B soit ajouté à l'ordonnancement σ si et seulement si son ratio $\frac{p(B)}{|B|}$ est inférieur à tous ceux des batches de σ. Afin de simplifier la présentation de l'algorithme de programmation dynamique, par la suite, ces conditions qui doivent êtres satisfaites pour chaque état.

L'ordonnancement partiel σ est soit dans l'état $(j, B_{i_1}, \ldots, B_{i_r})$, soit dans l'état $(j, \emptyset)$. Par la suite nous détaillons les décisions à prendre pour obtenir des ordonnancements dans chacun de ces états. Un ordonnancement dans l'état $(j, \emptyset)$ est obtenu suite à l'une des décisions suivantes :

- *Ajouter un batch contenant la tâche j à l'état* $(k, \emptyset)$. Le batch $B_{k-1} = \{j, \ldots, k-1\}$ est ajouté au début du meilleur ordonnancement d'état $(k, \emptyset)$.
- *Ajouter un batch g-complet* $B^*_{k'-1} = \{k, \ldots, k'-1\}$ *avec* $j < k$ *à l'état* $(k', B_{i_{\bar{1}}}, \ldots, B_{i_{\bar{r}}})$ où les batches $B_{i_{\bar{1}}}, \ldots, B_{i_{\bar{r}}}$ sont reportés par rapport au batch $B^*_{k'-1}$. Remarquons que ces batches $B_{i_{\bar{1}}}, \ldots, B_{i_{\bar{r}}}$ contiennent les tâches d'indices j à $k-1$, c'est-à-dire que $B_{i_{\bar{1}}} \cup \cdots \cup B_{i_{\bar{r}}} = \{j, \ldots, k-1\}$.

Un ordonnancement dans l'état $(j, B_{i_1}, \ldots, , B_{i_r})$ est obtenu suite à l'une des décisions suivantes :

- *Ajouter un batch* B_{i_1} *dont la tâche de tête est* i_1 *à l'état* $(j, B_{i_2}, \ldots, B_{i_r})$.
- *Ajouter un batch non reporté* $B_{k-1} = \{j, \ldots, k-1\}$ *à l'état* $(k, B_{i_1}, \ldots, B_{i_r})$. Remarquons qu'à l'état $(k, B_{i_1}, \ldots, B_{i_r})$ les batches $B_{i_1}, \ldots, B_{i_r}$ étaient reportés par rapport au batch B_{k-1}, ils sont désormais reportés également par rapport au batch B_{j-1}.
- *Ajouter un batch g-complet non reporté* $B^*_{k-1} = \{k', \ldots, k-1\}$ *avec* $j < k'$ *à l'état* $(k, B_{i_{\bar{1}}}, \ldots, B_{i_{\bar{r}}})$ tel que les batches $B_{i_{\bar{1}}}, \ldots, B_{i_{\bar{r}}}$ soient reportés par rapport au batch B^*_{k-1}. Remarquons que les batches $B_{i_{\bar{1}}}, \ldots, B_{i_{\bar{r}}}$ satisfont $(B_{i_{\bar{1}}}, \ldots, B_{i_{\bar{r}}}) \setminus (B_{i_1}, \ldots, B_{i_r}) = \{j, \ldots, k'-1\}$. En d'autres termes les batches reportés $B_{i_{\bar{1}}}, \ldots, B_{i_{\bar{r}}}$ sont divisés en 2 groupes. Le premier groupe contient les batches $B_{i_1}, \ldots, B_{i_r}$, ils sont composés de tâches d'indices inférieurs à $j-1$ et sont reportés par rapport aux batches B^*_{k-1} et B_{j-1}. Alors que les batches du second groupe sont composés des tâches d'indices j à $k'-1$ et sont reportés par rapport au batch B^*_{k-1} mais ne sont pas reportés par rapport au batch B_{j-1}.

Suite à cette présentation littéraire des décisions à prendre pour obtenir un ordonnancement σ dans l'état $(j, B_{i_1}, \ldots, B_{i_r})$ ou dans l'état $(j, \emptyset)$, nous les transformons en équations de récurrence afin de pouvoir automatiser le calcul.

Soit H_{k-1} l'ensemble des combinaisons réalisables de batches reportés par rapport au batch g-complet $B^*_{k-1} = \{k', \ldots, k-1\}$, c'est-à-dire l'ensemble des combinaisons de batches reportés $B_{i_1}, \ldots, B_{i_r}$ tels qu'il existe un ordonnancement dans l'état $(k, B_{i_1}, \ldots, B_{i_r})$ respectant les lemmes précédents. Soit H^j_{k-1} l'ensemble des combinaisons réalisables de batches reportés par

rapport au batch g-complet B^*_{k-1} et composés des tâches j à $k'-1$. Notons n' le nombre de tâches séquencées dans l'ordonnancement σ dans l'état $(j, B_{i_1}, \ldots, B_{i_r})$. Comme chaque batch $B_{i_k} = \{i'_k, \ldots, i_k\}$ est défini par les plus petit et plus grand indices de ses tâches, on obtient $n' = n - j + 1 + \sum_{k=1}^{r}(i_k - i'_k + 1)$.

Notons $F_j(B_{i_1}, \ldots, B_{i_r})$ et $F_j(\emptyset)$, respectivement, les sommes des dates de fin d'exécution du meilleur ordonnancement dans l'état $(j, B_{i_1}, \ldots, B_{i_r})$, pour $r \neq 0$, et $(j, \emptyset)$. Nous obtenons l'algorithme de programmation dynamique suivant :

- Initialisation : $F_{n+1}(\emptyset) = 0$
- Équations de récurrence :
  - Pour $j = n, n-1, \ldots, 1$:

$$F_j(\phi) = \min \begin{cases} \min_{k|(1+\alpha)a_{j-1} < a_{k-1} \leq (1+\alpha)a_j} \{F_k(\phi) + n'a_{k-1}\}, \\ \min_{B^*_{k'-1}=\{k,\ldots,k'-1\}|(B_{i_{\bar{1}}},\ldots,B_{i_{\bar{r}}})\in H^j_{k-1}} \{F_{k'}(B_{i_{\bar{1}}}, \ldots, B_{i_{\bar{r}}}) + n'a_{k'-1}\}, \end{cases}$$

  - Pour $j = n+1, n, \ldots, 1$, $r \leqslant r^*$ et pour tout $(B_{i_1}, \ldots, B_{i_r}) \in H_j$

$$F_j(B_{i_1}, \ldots, B_{i_r}) = \min \begin{cases} F_j(B_{i_2}, \ldots, B_{i_r}) + n'a_{i_1}, \\ \min_{k|(1+\alpha)a_{j-1} < a_{k-1} \leq (1+\alpha)a_j} \{F_k(B_1, \ldots, B_r) + n'a_{k-1}\}, \\ \min_{B^*_{k'-1}=\{k,\ldots,k'-1\}|(B_{i_{\bar{1}}},\ldots,B_{i_{\bar{r}}})\in H_{k'-1}} \{F_{k'}(B_{i_{\bar{1}}}, \ldots, B_{\bar{r}}) + \\ n'a_{k'-1}\}, \end{cases}$$

La valeur de la solution optimale est égale à $F_1(\emptyset)$, elle correspond à l'ordonnancement obtenu par "backtracking".

Nous considérons, finalement, la complexité de cet algorithme de programmation dynamique. Il y a $O(n^{2r^*+1})$ variables d'état $(j, B_{i_1}, \ldots, B_{i_r})$. Les deux termes de l'équation $F_j(\emptyset)$ sont calculés, respectivement, en $O(b^*)$ et $O(n^{2r^*})$. Quant-aux trois termes de l'équation de récurrence $F_j(B_{i_1}, \ldots, B_{i_r})$, ils sont calculés en temps constant pour le premier, $O(b^*)$ pour le second et $O(n^{2(r^*-r)})$ pour le troisième. C'est pourquoi la complexité générale de cet algorithme de programmation est $O(n^{2r^*})$, pour $O(n^{2r^*+1})$ variables. Pour rappel, d'après le lemme 5.3.3, r^* est inférieur à $(\frac{\log_2(b^*-1)-1}{\log_2 1/\alpha} + 2)\log_{1+\alpha} b^*$ lorsque $\alpha < 1$ et inférieur à $b^* \times \log_2 b^*$ lorsque $\alpha \geqslant 1$.

5.4 Minimiser des fonctions objectif avec dates de fin souhaitées

Dans cette section nous montrons tout d'abord la NP-complétude des problèmes de minimisation de critères réguliers basés sur des dates de fin souhaitées.

Théorème 5.4.1 *Le problème $B|G^\alpha_p = INT, \overline{d_j}|-$ est NP-complet même s'il n'y a que deux dates de fin impératives distinctes.*

Preuve. Nous démontrons ce résultat par une transformation polynomiale du problème NP-complet PARTITION (Garey et Johnson [83]) qui est le suivant : Soit $m+1$ entiers positifs notés $e_1, \ldots, e_m$ et E tels que $\sum_{j=1}^{m} e_j = 2E$, la question est la suivante : *Existe-t-il un ensemble $X \subset M = \{1, \ldots, m\}$ tel que $\sum_{j\in X} e_j = E$?*.

Sans perte de généralité, nous supposons les entiers indexés par ordre croissant $e_1 \leqslant \cdots \leqslant e_m$.

Soit une instance du problème PARTITION, nous construisons l'instance équivalente du problème $B|G_p = INT, \overline{d_j}|-$.

Afin de faciliter la preuve nous modifions légèrement le problème PARTITION à l'aide de la procédure suivante qui nous permet d'obtenir des nombres distincts. Fixer $e'_j = e_j,\ j = 1, \ldots, m$.

Pour chaque plus grand ensemble $\{i, i+1, \ldots, k\}$ tel que $e'_i = e'_{i+1} = \cdots = e'_k,\ 1 \leqslant i \leqslant k \leqslant m$, mettre à jour les valeurs tel que $e'_{j+1} = e'_j + \frac{1}{m^4},\ j = i, i+1, \ldots, k-1$. Nous obtenons alors $e_j \leqslant e'_j \leqslant e_j + \sum_{j=1}^{m} \frac{j}{m^4} \leqslant e_j + \frac{1}{m^2}$, ce qui implique $e'_j - \frac{1}{m^2} \leqslant e_j \leqslant e'_j,\ j = 1, \ldots, m$.

Soit $\sum_{j=1}^{m} e'_j = 2E'$, alors $2E \leqslant 2E' \leqslant 2E' \sum_{j=1}^{m} \frac{j}{m^4} \leqslant 2E + \frac{1}{m^2}$. Fixons $\alpha = \frac{1}{4m^4E'}$, $d = 2E' + 3\alpha E'$ et $D = 4E' + 5\alpha E'$.

L'instance du problème d'ordonnancement correspondant à cette instance du problème PARTITION est construite comme suit. Il y a $3m$ tâches. Notons les durées d'exécution et dates de fin souhaitées de la tâche j de type (t) $a_j^{(t)}$ et $d_j^{(t)}$, pour tout $j = 1, \ldots, m$. Ces valeurs sont définies comme suit :

- m tâches de type (1) telles que $a_j^{(1)} = e'_j,\ d_j^{(1)} = D,\ j = 1, \ldots, m$,
- m tâches de type $(1+\alpha)$ telles que $a_j^{(1+\alpha)} = (1+\alpha)e'_j,\ d_j^{(1+\alpha)} = d,\ j = 1, \ldots, m$,
- m tâches de type $(1+2\alpha)$ telles que $a_j^{(1+2\alpha)} = (1+2\alpha)e'_j,\ d_j^{(1+2\alpha)} = D,\ j = 1, \ldots, m$.

Ainsi définies, les tâches de type (1) ne sont pas compatibles entre elles, en effet

$$(1+\alpha)a_j^{(1)} = (1+\alpha)e'_j \leqslant e'_j + \frac{e'_j}{4m^4E'} < e'_j + \frac{1}{2m^4} < e'_{j+1} = a_{j+1}^{(1)},\ j = 1, \ldots, m-1$$

Comme $a_j^{(1+\alpha)} = (1+\alpha)a_j^{(1)}$ et $a_j^{(1+2\alpha)} = (1+2\alpha)a_j^{(1)}$ pour tout $j = 1, \ldots, m$, les tâches de type $(1+\alpha)$ sont incompatibles entre elles et les tâches de type $(1+2\alpha)$ sont incompatibles entre elles. De plus les tâches d'indices différents sont incompatibles car :

$$e'_j \leqslant (1+2\alpha)e'_j = e'_j + \frac{e'_j}{2m^4E'} < e'_j + \frac{1}{m^4} \leqslant e'_{j+1},\ j = 1, \ldots, m-1$$

Finalement les tâches de type (1) et de type $(1+2\alpha)$ de même indice sont incompatibles :

$$(1+\alpha)a_j^{(1)} = (1+\alpha)e'_j < (1+2\alpha)e'_j = a_j^{(1+2\alpha)}$$

Montrons désormais que le problème de PARTITION a une solution si et seulement s'il existe une solution réalisable de l'instance construite du problème d'ordonnancement. Cette preuve est accompagnée de la figure 5.1

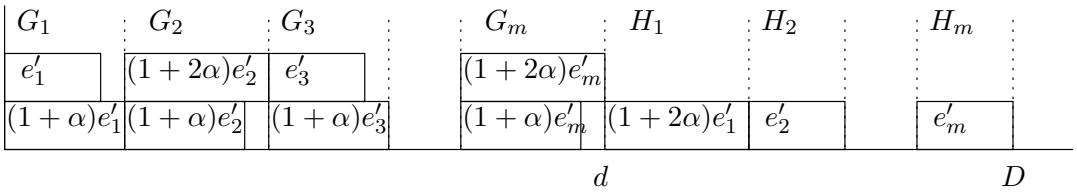


FIGURE 5.1 – Structure d'un ordonnancement réalisable

"Si". Supposons qu'il existe un ordonnancement réalisable de l'instance créée. Toutes les tâches de type $(1+\alpha)$ d'un tel ordonnancement doivent être terminées avant la date d, sachant que deux tâches de ce type ne peuvent appartenir au même batch car elles sont incompatibles deux à deux. Notons G_j le batch contenant la tâche d'indice j de type $(1+\alpha)$ pour tout $j = 1, \ldots, m$. La durée totale d'exécution des batches $G_1, \ldots, G_m$ est au minimum égale à $\sum_{j=1}^{m} a_j^{(1+\alpha)} = (1+\alpha)2E' = d - E'\alpha$. Ces inégalités impliquent qu'aucun autre batch ne soit

séquencé avant le dernier batch G_j car autrement la date de fin d'exécution de la dernière tâche de type $(1+\alpha)$ est supérieure à $d - E'\alpha + 1 > d$. Bien sûr l'ordre d'exécution des batches $G_1, \ldots, G_m$ n'a aucune importance, tout de même, nous supposons que les batches sont séquencés dans l'ordre $G_1, \ldots, G_m$.

Remarquons que seules les tâches de la paire $\{j^{(1+\alpha)}, j^{(1)}\}$ et les tâches de la paire $\{j^{(1+\alpha)}, j^{(1+2\alpha)}\}$ de même indice sont compatibles quel que soit $j = 1, \ldots, m$. Ainsi chaque batch G_j peut contenir au plus une tâche supplémentaire qui est soit la tâche $j^{(1)}$ soit la tâche $j^{(1+2\alpha)}$. C'est pourquoi il y a au moins m autres batches dans l'ordonnancement et au plus $2m$, notés $H_1, \ldots, H_k,\ m \leqslant k \leqslant 2m$. Ces batches succèdent au batch G_m et chacun de ces batches $H_i,\ 1 \leqslant i \leqslant k$ contient soit la tâche $j^{(1)}$ soit la tâche $j^{(1+2\alpha)}$, mais il ne peut contenir les deux, $j = 1, \ldots, m$. Sans perte de généralité nous supposons que ces batches sont séquencés dans l'ordre $H_1, \ldots, H_k$.

En omettant les contraintes liées aux dates de fin impératives, le batch H_m est complété au plus tôt lorsque les batches $G_j = \{j^{(1+\alpha)}, j^{(1+2\alpha)}\},\ H_j = \{j^{(1)}\},\ j = 1, \ldots, m$ sont complétés, cette date est égale à $C^0 = \sum_{j=1}^{m} a_j^{(1+2\alpha)} + \sum_{j=1}^{m} a_j^{(1)} = 4E' + 4\alpha E' = D - \alpha E'$. Ce qui nous permet de déduire qu'exactement m batches $H_1, \ldots, H_m$ succèdent au batch G_m, car autrement la date de fin de la dernière tâche est supérieure à $4E' + 4\alpha E' + 1 > D$ car $1 > \alpha E'$.

La figure 5.1 présente un exemple d'ordonnancement réalisable. Soit X l'ensemble d'indices de tâches de type $(1 + 2\alpha)$ ordonnancées dans les batches $G_1, \ldots, G_m$. Notons la date de fin d'exécution du batch G_m, $C(G_m)$. Afin de vérifier les contraintes de faisabilité relatives à la date de fin impérative d nous avons :

$$C(G_m) = \sum_{j \in M \setminus X} a_j^{(1+\alpha)} + \sum_{j \in X} a_j^{(1+2\alpha)} = 2(1+\alpha)E' + \alpha \sum_{j \in X} e'_j \leqslant d = 2E' + 3\alpha E'$$

Ainsi, $\sum_{j \in X} e'_j \leqslant E'$.

Calculons maintenant la durée d'exécution totale des batches $H_1, \ldots, H_m$ notée $P(H)$:

$$P(H) = \sum_{j \in M \setminus X} a_j^{(1+2\alpha)} + \sum_{j \in X} a_j^{(1)} = 2E' + 4\alpha E' - 2\alpha \sum_{j \in X} e'_j$$

Pour respecter les contraintes de faisabilité liées à la date de fin impérative D, nous devons respecter :

$$C(G_m) + P(H) = 4E' + 6\alpha E' - \sum_{j \in X} e'_j \leqslant 4E' + 5\alpha E'$$

Ainsi $\sum_{j \in X} e'_j \geqslant E'$, c'est pourquoi $\sum_{j \in X} e'_j = E'$. Cette égalité implique les relations $2E \leqslant 2E' \leqslant 2E + \frac{1}{m^2}$ et $e'_j - \frac{1}{m^2} \leqslant e_j \leqslant e'_j,\ j = 1, \ldots, m$, d'où

$$E - 1 < E - \frac{1}{m} \leqslant E' - \frac{1}{m} \leqslant \sum_{j \in X} e_j \leqslant E' \leqslant E + \frac{1}{2m^2} < E + 1$$

Comme toutes les valeurs de e_j sont entières, $\sum_{j \in X} e_j = E$ est satisfait dans le cas où la solution est réalisable, c'est-à-dire que X est une solution du problème PARTITION.

"Seulement si". Supposons qu'il existe une solution du problème PARTITION, notée X, alors il existe un ensemble $X \subset N = \{1, \ldots, m\}$ tel que $\sum_{j \in X} e_j = E$. Sans perte de généralité nous supposons les valeurs de e_j distinctes, en utilisant la même transformation que dans la partie précédente. Nous transformons cette solution en une instance du problème d'ordonnancement décrit dans la partie "Si". Soit X l'ensemble des indices de tâches de types $(1+2\alpha)$ ordonnancées

dans les batches $G_1, \ldots, G_m$. Tout d'abord vérifions que le batch G_m est complété à temps. Nous obtenons alors :

$$\begin{aligned} C(G_m) &= \sum_{j \in M \setminus X} a_j^{(1+\alpha)} + \sum_{j \in X} a_j^{(1+2\alpha)} = \sum_{j \in M \setminus X} (1+\alpha)e_j + \sum_{j \in X} (1+2\alpha)e_j \\ &= (1+\alpha) \sum_{j \in M} e_j + \alpha \sum_{j \in X} e_j = (1+\alpha)2E + \alpha E = d \end{aligned}$$

Ainsi toutes les tâches de type $(1+\alpha)$ sont bien complétées à la date d. Calculons maintenant la durée totale des batches $H_1, \ldots, H_m$ notée $P(H)$, pour ce faire remarquons que $\sum_{j \in M \setminus X} e_j = 2E - \sum_{j \in X} e_j = E$:

$$\begin{aligned} P(H) &= \sum_{j \in M \setminus X} a_j^{(1+2\epsilon)} + \sum_{j \in X} a_j^{(1)} = \sum_{j \in M \setminus X} (1+2\alpha)e_j + \sum_{j \in X} e_j \\ &= \sum_{j \in M} e_j + 2\alpha \sum_{j \in M \setminus X} e_j = 2E + 2\alpha E \end{aligned}$$

Ainsi la date de fin d'exécution du batch H_m est la suivante :

$$C(G_m) + P(H) = 4E + 5\alpha E = D$$

La solution du problème PARTITION correspond bien a un ordonnancement réalisable. Ce qui complète la preuve.

□

Corollaire 5.4.2 *Le problème* $B|G_p^\alpha = INT|\gamma$ *est NP-difficile pour* $\gamma \in \{L_{max},\ T_{max},\ f_{max},\ \sum(w_j)U_j,\ \sum(w_j)T_j\}$.

5.4.1 Algorithme de programmation dynamique

Dans cette section nous développons un algorithme de programmation dynamique pour les problèmes de minimisation des fonctions objectifs régulières avec dates de fin souhaitées. Supposons que toutes les durées minimales d'exécution a_i soient positives et indexées par ordre croissant, $i = 1, \ldots, n$, avec $A = \sum_{i=1}^{n} a_i$. Soit $X = \{a_i\}_{i=1,\ldots,n} \cup \{0\} \cup \{(1+\alpha)a_n\} = \{x_1, \ldots, x_{n+2}\}$ un ensemble de points initiaux des intervalles de durées d'exécution des tâches.

A. Minimiser le retard algébrique

Notons par $P_{\cup_{k=1}^{l}(x_{i_{2k-1}}, x_{i_{2k}})}$ le sous-problème de minimisation du retard algébrique restreint aux tâches dont l'intervalle de durées d'exécution est inclus (dans sa totalité) dans l'un des intervalles ouverts $]x_{i_{2k-1}}, x_{i_{2k}}[,\ x_{i_{2k-1}} < x_{i_{2k}},\ k = 1, \ldots, l$. Sans perte de généralité nous supposons $x_{i_{2k-1}} \leqslant x_{i_{2k}}$.

Soit t la date de disponibilité de la machine, $F_{\cup_{k=1}^{l}(x_{i_{2k-1}}, x_{i_{2k}})}(t)$ indique la valeur optimale de l'objectif pour le problème restreint $P_{\cup_{k=1}^{l}(x_{i_{2k-1}}, x_{i_{2k}})}$, lorsque la machine est disponible à la date t. Si aucune tâche n'est incluse dans l'intervalle $]x_i, x_j[$ alors $F_{\cup_{k=1}^{l}(x_{i_{2k-1}}, x_{i_{2k}}) \cup (x_i, x_j)}(t) = F_{\cup_{k=1}^{l}(x_{i_{2k-1}}, x_{i_{2k}})}(t)$, et $F_\emptyset(t) = 0$.

Notre algorithme de programmation dynamique est basé sur le lemme 5.4.3, ce lemme permet de séparer un problème $P_{\cup_{k=1}^{l}(x_{i_{2k-1}},x_{i_{2k}})}$ en un nombre fini de sous-problèmes. Soit $k = argmin\{d_j | j \in P_{\cup_{k=1}^{l}(x_{i_{2k-1}},x_{i_{2k}})}\}$ l'indice de la tâche de plus petite date de fin impérative du sous-problème $P_{\cup_{k=1}^{l}(x_{i_{2k-1}},x_{i_{2k}})}$. Sans perte de généralité nous supposons que la paire (x_{i_1}, x_{i_2}) est la paire de valeurs de X contenant la tâche k.

Lemme 5.4.3 *Il existe un ordonnancement optimal du problème $P_{\cup_{k=1}^{l}(x_{i_{2k-1}},x_{i_{2k}})}$ pour lequel le batch qui contient la tâche k est séquencé en première position.*

Preuve. Soit $\sigma = (B_1, \ldots, B_l)$ un ordonnancement réalisable dans lequel $k \in B_j$. Soit $\sigma' = (B_1, \ldots, B_j, B_{j-1}, \ldots, B_l)$ l'ordonnancement obtenu en échangeant les batches B_j et B_{j-1} dans σ. Ainsi σ' est réalisable, de plus les dates de fin d'exécution de tous les batches, exceptées celles de B_j et B_{j-1}, n'ont pas été modifiées. D'où :

$$L_{max}(\sigma') - L_{max}(\sigma) =$$
$$\max\{L_{max}(\sigma \setminus \{B_{j-1}, B_j\}), C + p(B_j) - d_k, C + p(B_j) + p(B_{j-1}) - d_{B_{j-1}}\}$$
$$- \max\{L_{max}(\sigma \setminus \{B_{j-1}, B_j\}), C + p(B_j) + p(B_{j-1}) - d_k, C + p(B_{j-1}) - d_{B_{j-1}}\}$$

où C est la date de début d'exécution du batch B_{j-1} dans l'ordonnancement σ, $d_{B_{j-1}}$ la plus petite date de fin d'exécution souhaitée des tâches du batch B_{j-1}, et $L_{max}(\sigma \setminus \{B_{j-1}, B_j\})$ le retard maximum des tâches des batches de $\sigma \setminus \{B_{j-1}, B_j\}$.

Comme $d_k \leqslant d_{B_{j-1}}$ et $p(B_{j-1}) \geqslant 0$ on a $L_{max}(\sigma') \leqslant L_{max}(\sigma)$. Un nombre fini de répétition de cet échange permet d'obtenir un ordonnancement optimal respectant le lemme.

□

Soit $Y = X \cap [a_k, (1+\alpha)a_k]$ l'ensemble de points initiaux de X inclus dans l'intervalle $[a_k, (1+\alpha)a_k]$, l'algorithme de programmation dynamique est le suivant :

Initialisation :

$$F_{\emptyset}(t) = 0 \quad \forall t = a_1, \ldots, A$$

Équation de récurrence :

$$F_{\cup_{j=1}^{l}(x_{i_{2j-1}},x_{i_{2j}})}(t) = \min_{z \in Y}\{\max\{F_{\cup_{j=2}^{l}(x_{i_{2j-1}},x_{i_{2j}})\cup(x_{i_1},z)\cup(z,x_{i_2})}(t+z), t+z-d_k\}\}$$

La valeur optimale : $L_{max} = F_{(x_1,x_{n+2})}(0)$.

Quel que soit le sous-problème $P_{\cup_{j=1}^{l}(x_{i_{2j-1}},x_{i_{2j}})}$, un intervalle $(x_{i_{2j-1}}, x_{i_{2j}}),\ j = 1, \ldots, l$ n'est pas vide si et seulement si $(1+\alpha)x_{i_{2j-1}} < x_{i_{2j}}$, de plus $x_{i_{2j}} \leqslant x_{i_{2j+1}}$ d'où :

$$(1+\alpha)x_{i_{2j-1}} < x_{i_{2j}} \leqslant x_{i_{2j+1}} \quad \forall j = 1, \ldots, l-1$$

Ainsi,

$$(1+\alpha)^l x_1 < x_{n+2}$$

Étant donné que $1 \leqslant x_1$, on a $l \leqslant \log_{1+\alpha} \alpha_n$. C'est pourquoi le nombre d'intervalles non vides dans le sous-problème $P_{\cup_{j=1}^{l}(x_{i_{2j-1}},x_{i_{2j}})}$ est inférieur à $\log_{1+\alpha} a_n$. De plus chaque intervalle $(x_{i_{2j-1}}, x_{i_{2j}})$ est défini par deux points initiaux de l'ensemble X, il y a donc au plus $n^{2\log_{1+\alpha} a_n}$ combinaisons d'intervalles $(x_{i_{2j-1}}, x_{i_{2j}})$. Ainsi pour chaque valeur de $t,\ t = a_1, \ldots, A$ nous devons calculer au plus $n^{2\log_{1+\alpha} a_n}$ valeurs de la fonction objectif $F_{\cup_{j=1}^{l}(x_{i_{2j-1}},x_{i_{2j}})}(t)$. Comme chaque équation de récurrence est calculé en $O(n)$, la complexité de l'algorithme de programmation dynamique est de $O(n^{1+2\log_{1+\alpha} a_n} \times A)$. Nous sommes conscients que la complexité de cet algorithme n'est pas intéressante lorsque α est inférieur à 1, et travaillons actuellement sur ce problème.

B. Minimiser la somme pondérée des retards

Nous utilisons ici les mêmes notations que pour la minimisation du retard algébrique. Ainsi, $P_{\cup_{j=1}^{l}(x_{i_{2j-1}},x_{i_{2j}})}$ représente le sous-problème de minimisation de la somme pondérée des retards restreint aux tâches dont l'intervalle de durées d'exécution est inclus dans l'un des intervalles ouverts $]x_{i_{2j-1}}, x_{i_{2j}}[,\ j = 1, \ldots, l$. La date de disponibilité de la machine est notée t, ainsi la valeur optimale de la fonction objectif pour le problème $P_{\cup_{j=1}^{l}(x_{i_{2j-1}},x_{i_{2j}})}$ dont la date de fin d'exécution est égale à t est notée $F_{\cup_{j=1}^{l}(x_{i_{2j-1}},x_{i_{2j}})}(t)$. Si aucune tâche n'est incluse dans l'intervalle $]x_i, x_j[$ alors $F_{\cup_{k=1}^{l}(x_{i_{2k-1}},x_{i_{2k}})\cup(x_i,x_j)}(t) = F_{\cup_{k=1}^{l}(x_{i_{2k-1}},x_{i_{2k}})}(t)$, et $F_{\emptyset}(t) = 0$.

Contrairement au problème de minimisation du retard algébrique, nous ne pouvons pas restreindre notre choix aux ordonnancements séquençant le batch contenant la tâche de plus petite date de fin souhaitée en première position. Nous devons donc explorer toutes les possibilités pour le premier batch, c'est-à-dire dans tous les intervalles $]x_{i_{2j-1}}, x_{i_{2j}}[,\ j = 1, \ldots, l$.

Soit j_z l'indice de l'intervalle $(x_{i_{2j_z-1}}, x_{i_{2j_z}})$ qui contient la durée d'exécution p_z, et B le batch de durée d'exécution p_z qui contient toutes les tâches de l'intervalle $(x_{i_{2j_z-1}}, x_{i_{2j_z}})$ compatibles avec B. L'algorithme de programmation dynamique est le suivant :

Initialisation :

$$F_{\emptyset}(t) = 0 \quad \forall t = a_n, \ldots, \sum_{i=1}^{n} x_i$$

Équation de récurrence :

$$F_{\cup_{j=1}^{l}(x_{i_{2j-1}},x_{i_{2j}})}(t) = \min_{p_z \in \cup_{j=1}^{l}(x_{i_{2j-1}},x_{i_{2j}})} \{F_{\cup_{j\neq j_z}(x_{i_{2j-1}},x_{i_{2j}})\cup(x_{i_{2j_z-1}},p_z)\cup(p_z,x_{i_{2j_z}})}(t+z) + \sum_{j\in B} w_j \max\{t + p_z - d_j, 0\}\}$$

La valeur optimale : $\sum w_j T_j = F_{(x_1,x_{n+2})}(0)$.

Dans cet algorithme de programmation dynamique, chaque équation de récurrence est calculée en $O(n^2)$ contre $O(n)$ pour la minimisation du retard algébrique, c'est pourquoi la complexité de l'algorithme de programmation dynamique est $O(n^{2+2\log_{1+\alpha} a_n} \times A)$. Nous sommes conscients que la complexité de cet algorithme n'est pas intéressante lorsque α est inférieur à 1, et travaillons actuellement sur ce problème.

5.5 Conclusion du chapitre

Dans ce chapitre nous avons montré la complexité des différents problèmes de minimisation de critères réguliers sur une machine à traitement par batches avec compatibilité entre les tâches et capacité infinie. Ainsi nous présentons un algorithme de programmation dynamique pour la minimisation de la somme des dates de fin ($B|G_p^{\alpha} = INT|\sum C_j$) dans la section 5.3. Dans la section 5.4 nous montrons la NP-complétude des problèmes de minimisation de critères réguliers avec date de fin souhaitée, ensuite nous présentons des algorithmes de programmation dynamique permettant de résoudre les problèmes $B|G_p^U = INT|L_{max}$ et $B|G_p^U = INT|\sum w_j T_j$.

Nous sommes actuellement en cours de rédaction d'un article présentant les résultats de ce chapitre.

Chapitre 6

Conclusion et perspectives

Dans la première section de ce dernier chapitre, nous rappelons les résultats obtenus dans cette thèse. La seconde section de ce chapitre est consacrée aux perspectives résultant des travaux présentés ici.

Sommaire

6.1 Conclusion 120
6.2 Perspectives 121
6.2.1 Intensification des expériences 121
6.2.2 Extension des modèles étudiés 121
6.2.3 Autres perspectives 122

6.1 Conclusion

Dans cette thèse nous avons étudié les problèmes d'ordonnancement de machines à traitement par batches avec compatibilité entre les tâches. La compatibilité entre les tâches est décrite par les intervalles de durées d'exécution des tâches.

Le premier chapitre de cette thèse présente, après une introduction des problématiques d'ordonnancement de la production, les problèmes d'ordonnancement étudiés dans ce manuscrit, ainsi qu'une application industrielle de ceux-ci.

Le second chapitre décrit les résultats existant dans la littérature pour les problèmes de minimisation de critères réguliers, sur des machines à traitement par batches avec, et sans, contraintes de compatibilité. Ce chapitre décrit aussi l'étude des problèmes de flowshops hybrides composés de machines disjonctives et/ou à traitement par batches. L'état-de-l'art nous a permis de constater que les problèmes traités ici ont été très peu considérés dans la littérature. En effet, seul le problème de minimisation de la date de fin de l'ordonnancement d'une machine à traitement par batches et compatibilité entre les tâches a été étudié par Oulamara et al. [176]. Les problèmes de flowshops hybrides avec machines à traitement par batches et compatibilité entre les tâches n'ont pas été étudiés jusqu'à présent.

Dans le troisième chapitre, nous avons développé des méthodes approchées pour les problèmes de minimisation du makespan de flowshops hybrides avec machines à traitement par batches et compatibilité entre les tâches. Ainsi, nous avons développé 6 heuristiques constructives à performances garanties pour le problème général $(HF2(m_1, Bm_2)|G_p = INT, b = k|C_{max})$. Heuristiques que nous avons adaptées par la suite aux problèmes avec une seule machine sur l'un des étages $(HF2(1, Bm)|G_p = INT, b = k|C_{max}$ et $HF2(m, B)|G_p = INT, b = k|C_{max})$. Les expériences que nous avons menées indiquent que ces heuristiques fournissent très rapidement des solutions proches de l'optimum (en moyenne 1% de la borne inférieure) pour des instances de 200 tâches. Nous avons aussi développé un schéma d'approximation polynomial (PTAS) pour le problème de flowshop hybride avec machines à traitement par batches, compatibilité entre les tâches et durées d'exécution du premier étage identiques $(HF2(m_1, Bm_2)|G_p^U = INT, b = k, p_j = p|C_{max})$. Cette méthode fournit, en temps polynomial par rapport à la taille de l'instance, une solution dont l'écart par rapport à la solution optimale est inférieur à $O(\epsilon)$.

Dans le quatrième chapitre nous avons développé deux méthodes exactes de type procédure par séparation et évaluation (PSE). La première méthode énumère les ordonnancements non dominés du premier étage, puis énumère, pour chacun de ces ordonnancements, les ordonnancements du second étage respectant les solutions obtenues sur le premier étage. La deuxième méthode effectue des énumérations comparables pour le problème inverse (inversion des étages). Ainsi, dans un premier temps, tous les ordonnancements non dominés, de l'étage composé des machines à traitement par batches, sont énumérés puis, pour chacun de ces ordonnancements, l'ordonnancement optimal de l'étage composé des machines disjonctives, est recherché. Cette deuxième étape est calculée à l'aide de deux méthodes différentes : une première méthode, que nous avons développée, utilise un algorithme de programmation dynamique, alors que la seconde méthode, développée par Gharbi et Haouari [84], utilise leur méthode. Les expériences menées indiquent que la méthode inverse utilisant la programmation dynamique fournit la solution optimale, en moins de 10 minutes, pour une partie des instances de 20 tâches.

Finalement, le dernier chapitre traite des problèmes de minimisation de critères réguliers sur une seule machine à traitement par batches, compatibilité entre les tâches et capacité infinie. Nous avons développé un algorithme de programmation dynamique pour le problème de minimisation de la somme des dates de fin d'exécution. Et dans la dernière section nous démontrons la NP-complétude des problèmes de minimisation de critères à base de dates de fin souhaitée, avant de développer deux algorithmes de programmation dynamique pour les problèmes de minimisation du plus grand retard et de la somme pondérée des retards.

6.2 Perspectives

Les perspectives liées à ces travaux sont multiples. Dans la sous-section 6.2.1 nous présentons les intensifications des résultats expérimentaux qui permettraient de mieux estimer l'efficacité des méthodes que nous avons développées. Ensuite, dans la sous-section 6.2.2, nous présentons quelques problèmes dont il serait intéressant d'étudier la complexité, et de fournir quelques méthodes de résolution. La dernière sous-section 6.2.3 présente des perspectives visant à étendre les domaines d'application des résultats présentés ici.

6.2.1 Intensification des expériences

Tout d'abord, nous n'avons pas effectué d'expérimentations permettant de mesurer l'intérêt du schéma d'approximation polynomial. C'est pourquoi nous souhaitons, dans un avenir proche, programmer ce schéma d'approximation polynomial afin de mesurer son efficacité. Il sera, par exemple, intéressant d'observer le temps d'exécution moyen nécessaire pour obtenir une solution inférieure à la meilleure solution fournie par une heuristique. Si le schéma d'approximation peut fournir rapidement des solutions avec une erreur relative faible (inférieure à celles des heuristiques), alors nous pourrions utiliser ce PTAS pour obtenir de bonnes solutions admissibles, lorsque les durées d'exécution sur le premier étage sont relativement proches. Notons que, dans le cas industriel qui nous a inspiré, l'écart type entre les durées d'exécution du premier étage est faible.

Les expérimentations concernant les méthodes exactes sont également à intensifier. Ainsi il serait intéressant de tester les méthodes exactes lorsque plusieurs tâches ont les mêmes durées sur les deux étages. En effet, alors que nous avons développé des règles de dominance, permettant de limiter l'augmentation du nombre de nœuds à explorer en raison de l'existence de tâches identiques, nous n'avons pas testé leur efficacité. C'est pourquoi il conviendrait de créer des instances particulières pour ces tests.

6.2.2 Extension des modèles étudiés

Une première piste de travail serait d'adapter les différents résultats, de cette thèse, qui concernent les flowshops hybrides avec machines classiques au premier étage, et machines à traitement par batches au second étage, au cas plus général, pour lequel les deux étages sont composés de machines à traitement par batches et intervalles de durées d'exécution compatibles. Cette extension correspond finalement à deux problèmes différents. En effet, hormis certains cas particuliers, pour lesquels les contraintes de compatibilité sont les mêmes sur les deux étages, elles diffèrent généralement d'un étage à l'autre. Il y a alors deux types de problèmes : soit les batches doivent être identiques sur le premier et le second étage (propriété de consistance des batches), et dans ce cas les tâches d'un batch doivent être compatibles sur les deux étages,

soit les batches peuvent être différents d'un étage à l'autre. Dans le premier cas, lorsque les batches sont créés ils peuvent être considérés comme les tâches d'un flowshop hybride avec machines disjonctives, c'est pourquoi nous nous intéressons ici au cas où les batches peuvent être différents d'un étage à l'autre. L'adaptation des heuristiques, qui semble intéressante et relativement aisée à mettre en place, est de créer une liste de batches sur chaque étage, puis de trier ces deux listes avec des règles qui restent à développer, LPBT et Johnson ne semblant pas adaptées. Cependant nous risquons d'utiliser soit la règle LPT, soit une règle de liste pour trier les batches, donc les garanties de performances devraient être équivalentes à celles développées ici. De plus, ces problèmes semblent plus adaptés à l'obtention d'instances atteignant les garanties de performances développées dans cette thèse. Finalement, l'adaptation des méthodes exactes soulèvera plus de difficultés. Les étages étant désormais tous deux composés de machines à traitement par batch, une seule méthode subsistera. Cette méthode combinera la première étape de la méthode inverse avec la seconde étape de la méthode directe. Il faudra néanmoins effectuer quelques modifications sur les bornes et les règles de dominance. Par manque de temps nous n'avons pas rédigé ces méthodes et effectué les expériences nécessaires à une publication. L'étude de ces expérimentations serait très intéressante : en effet il y a peu de chances pour que les batches soient composés de la même façon sur le premier et sur le second étage ; l'ordre des batches sur chaque étage devient alors difficile à définir mais est primordial.

Dans le chapitre 5, nous avons étudié la complexité des problèmes de minimisation de critères réguliers sur une machine à traitement par batches à capacité infinie et compatibilité entre les tâches. Dans ce cas, seuls deux problèmes ne sont pas démontrés NP-difficiles, or ces problèmes sont des cas particuliers des problèmes de minimisation de critères réguliers sur une machine à traitement par batches de capacité finie et compatibilité entre les tâches (cas où $b = n$). Donc les problèmes avec capacité finie sont au moins aussi difficiles. Ce qui est confirmé par le tableau 2.3 de la section 2.2.1 qui présente la complexité des problèmes de type $B|b = k|\gamma$, effectivement dans ce tableau seuls les problèmes de minimisation du makespan et de la somme (pondérée) des dates de fin de l'ordonnancement ne sont pas NP-difficiles au sens fort. Lorsque l'on ajoute les contraintes de compatibilité entre les tâches, le problème $B|G_p = INT, b = k|C_{max}$ est résolu polynomialement par l'algorithme FCBLPT. Pour ce qui est de la minimisation de la somme des durées d'exécution, l'algorithme de programmation dynamique développé dans la section 5.3, pour le problème avec capacité infinie, devrait être adaptable sans trop de changements à ce nouveau problème. C'est pourquoi la minimisation des problèmes de somme de durées d'exécution pondérée ou non ne semblent pas NP-difficile au sens fort, alors que les problèmes L_{max}, $\sum T_j$ et $\sum U_j$ le sont.

Suite à ces premières évolutions, dont les développements potentiels sont relativement immédiats si nos conjectures intuitives sont avérées, il pourra également être intéressant d'étendre l'étude des problèmes de minimisation de critères réguliers sur des machines à traitement par batches et intervalles de durées d'exécution compatibles à des structures d'atelier plus complexes de type machines parallèles, flowshops (hybrides), jobshops et openshops. Ces ateliers peuvent comprendre des étages avec machines disjonctives, et les compatibilités peuvent différer d'un étage à l'autre.

6.2.3 Autres perspectives

Nous distinguons une première série de perspectives, portant sur une représentation approfondie du problème industriel qui nous a inspiré. Dans le chapitre liée aux aspects applicatifs des

problèmes de machines à traitement par batches, nous avons supposé que toutes les machines étaient identiques et qu'il n'y avait pas de temps de réglages. Cependant le problème industriel est nettement plus complexe ; comme nous l'avons expliqué en présentant le problème industriel (section 1.4), la forme commerciale des pneus est définie par le moule utilisé lors de la cuisson. Donc, lorsque deux types de pneus différents sont cuits consécutivement dans un même emplacement, il est nécessaire d'effectuer un changement de moule. Or ce changement de moule est relativement long (de 45 minutes à plusieurs heures suivant la taille du moule), et cette opération nécessite des opérations humaines, effectuées par des ouvriers spécialisés qui sont en nombre limité dans l'usine.

La seule prise en compte des temps de réglages, induits par ces changements de moules, rend les méthodes que nous avons développées inapplicables. En effet, les heuristiques n'optimiserons pas les temps de réglage et les bornes inférieures ne les prendront pas en compte. Il est donc nécessaire de développer d'autres méthodes de résolution, qui peuvent être, par exemple, des algorithmes génétiques.

Lorsque l'on prend également en compte le partage entre machines d'équipes spécialisées dans le changement de moules, le problème devient encore plus complexe. En effet il faut désormais faire en sorte qu'à chaque unité de temps le nombre de changements de moules soit inférieur au nombre d'équipes spécialisées. À première vue, ce nouveau problème semble combiner les problèmes d'ordonnancement de flowshop hybride avec machines à traitement par batches et des problèmes de ressources cumulatives secondaires.

En ce qui concerne les incompatibilités entre les tâches, nous avons réduit le problème aux incompatibilités temporelles, mais il peut également exister des incompatibilités entre des types de pneus d'intervalles de durées d'exécution compatibles. En effet, en pratique deux pneus compatibles selon leurs durées de cuisson peuvent ne pas être compatibles, par exemple, si la température maximale utilisée en fin d'opération n'est pas la même. Alors que ces nouvelles incompatibilités perturbent les algorithmes de programmation dynamique et le schéma d'approximation polynomial, leur prise en compte dans le cadre des heuristiques et des méthodes exactes ne posera pas de problèmes. Notons que l'incompatibilité entre les moules est équivalente à une incompatibilité entre les pneus.

Nous aurions pu également ajouter les contraintes liées au nombre limité de moules, les incompatibilités entre certains types de pneus et certaines machines du premier et du second étage (taille du pneu excessive).

Les résultats fournis par les heuristiques sur de grandes instances (200 tâches) du problème $HF2(m_1, Bm_2)|G_p = INT, b = k|C_{max}$ étant proches de la borne inférieure (en moyenne 1%), nous n'avons pas estimé intéressant de développer des méta-heuristiques pour ce problème. En revanche, nous souhaitions développer des méta-heuristiques pour un problème plus proche de celui rencontré dans l'industrie, en utilisant des ordres de grandeurs issus de l'industrie (nombres de tâches identiques, durées...) et des contraintes industrielles perturbant l'ordonnancement. Cependant, le partenaire industriel ne disposant pas de moyens humains nécessaires à cette définition du problème, nous n'avons pu définir correctement ce problème industriel. C'est pourquoi nous nous sommes orienté vers des pistes de recherches plus théoriques.

Les applications industrielles qui utilisent des machines à traitement par batches et intervalles de durées d'exécution compatibles sont nombreuses, ces applications concernent généralement des applications de trempage ou de cuisson. Une seconde différenciation porte sur les limites de capacité des batches. En effet, dans certains problèmes toutes les tâches ont le même volume, alors que dans d'autres problèmes les volumes sont variables, les limites de capacités peuvent,

également, être plus complexes.

Par exemple dans la cristallerie, afin d'opacifier le verre, les pièces sont trempées par lots dans des bains d'acides. La durée de trempage de ces pièces doit appartenir à un intervalle spécifique à chaque type de pièce, d'où les contraintes de compatibilité de type intervalles de durées d'exécution compatibles. Les pièces sont posées sur des plaques en caoutchouc contenant plusieurs casiers de différentes tailles. Notons qu'il existe plusieurs agencements différents et que plusieurs plaques sont trempées simultanément dans un même bac. C'est pourquoi, nous pouvons considérer que la création de batches nécessite de résoudre des problèmes de découpe $2D^+$ afin d'optimiser la taille des batches.

Cette combinaison de batches et de problèmes de découpes est également présente dans les fours, par exemple lors de la cuisson de faïence ou de porcelaine. De tels problèmes sont également présents dans la sidérurgie, dans les cloches de recuit de bobines d'acier ou dans les fours de réchauffage des brames avant laminage à chaud. Dans ces cas, il existe généralement de nombreuses contraintes de placement lors de la composition des batches.

Le dernier exemple que nous présentons est le cas de la galvanoplastie qui permet d'appliquer un dépôt métallique à l'aide d'un courant électrique. Les pièces sont trempées dans des cuves à l'aide de supports qui contiennent généralement plusieurs pièces. Chaque support trempe et ressort les pièces, qu'il contient en même temps. De plus, d'une cuve à l'autre les pièces ne peuvent être enlevées du support. Comme la durée, pendant laquelle chaque pièce est trempée, appartient à un intervalle dépendant du type de pièces, les pièces posées sur un même support doivent avoir une durée de trempe commune pour chaque étage. Nous sommes donc en présence d'un flowshop (hybride ou non), composé de machines à traitement par batches avec intervalles de durées d'exécution compatibles, et les batches doivent être conservés d'un étage sur l'autre. De plus les pièces ne peuvent être exposées trop longtemps à l'air libre entre deux étages, c'est-à-dire que les temps morts ne sont pas acceptés. En supposant qu'il existe plusieurs bacs pour chaque étage nous avons donc un flowshop hybride sans temps morts avec machines à traitement par batches sur tous les étages et intervalles de durées d'exécution compatibles. Ces problèmes sont généralement traités par les techniques de hoist-scheduling qui supposent que les batches soient créés avant l'ordonnancement des différents supports.

Les perspectives évoquées ici, nous permettent de remarquer que les applications industrielles utilisant ces contraintes de compatibilité sont diverses et variées. De plus, elles fournissent des problèmes théoriques intéressants, c'est pourquoi nous espérons que cette thèse sera suivie par d'autres travaux sur le sujet.

Annexe A

Résultats expérimentaux détaillés des heuristiques.

m_1-m_2	k	α	H_{LPT}						H_J						H_{LBPT}						H^*	
			Normal			Inverse			Normal			Inverse			Normal			Inverse				
			Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max
1–1	2	0,1	49,68	51,89	0	1,16	1,92	0	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01
		1	50,09	54,44	0	1,14	1,53	0	0,01	0,02	10	0,01	0,02	10	0,01	0,02	10	0,01	0,02	10	0,01	0,02
	3	0,1	33,38	36,52	0	0,85	0,98	0	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01
		1	33,33	35,93	0	0,86	1,03	0	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01
	10	0,1	11,49	12,05	0	0,84	0,88	0	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01
		1	10,34	10,97	0	0,84	0,88	0	0,04	0,06	10	0,04	0,06	10	0,04	0,06	10	0,04	0,06	10	0,04	0,06
1–10	2	0,1	0,87	1,10	0	0,85	0,91	0	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01
		1	0,96	1,70	0	0,85	0,89	0	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01
	3	0,1	0,84	0,87	0	0,84	0,87	0	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01
		1	0,85	0,90	0	0,85	0,90	0	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01
	10	0,1	0,83	0,86	0	0,83	0,86	0	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01
		1	0,85	0,89	0	0,85	0,89	0	0,04	0,05	10	0,04	0,05	10	0,04	0,05	10	0,04	0,05	10	0,04	0,05
2–2	2	0,1	51,17	59,43	0	2,45	3,31	0	0,38	0,69	1	0,11	0,21	8	0,38	0,70	0	0,41	0,70	1	0,09	0,16
		1	50,84	54,97	0	2,46	3,71	0	0,34	0,81	2	0,09	0,23	6	0,34	0,78	2	0,26	0,60	2	0,07	0,16
	3	0,1	33,14	34,19	0	2,15	2,64	0	0,45	0,82	3	0,30	0,67	7	0,44	0,81	1	0,46	0,67	4	0,27	0,58
		1	32,71	34,53	0	2,06	2,42	0	0,38	0,79	2	0,24	0,45	6	0,38	0,79	2	0,40	0,87	2	0,22	0,44
	10	0,1	8,96	10,10	0	2,05	2,31	0	0,40	0,65	2	0,23	0,48	8	0,40	0,65	2	0,23	0,48	8	0,21	0,48
		1	7,90	9,21	0	1,93	2,17	0	0,30	0,56	7	0,55	0,82	3	0,30	0,56	7	0,55	0,82	3	0,28	0,56
2–5	2	0,1	18,05	20,38	0	1,95	2,30	0	0,27	0,63	4	0,26	0,75	3	0,28	0,63	4	0,31	0,75	5	0,12	0,23
		1	18,26	20,31	0	2,00	2,23	0	0,31	0,53	4	0,29	0,56	6	0,31	0,51	3	0,41	0,70	2	0,18	0,28
	3	0,1	10,65	11,44	0	2,04	2,56	0	0,40	0,87	3	0,40	0,75	5	0,38	0,87	5	0,40	0,75	5	0,24	0,46
		1	10,72	11,87	0	2,20	2,63	0	0,51	0,79	3	0,29	0,70	7	0,51	0,79	3	0,29	0,70	7	0,21	0,44
	10	0,1	1,92	2,42	0	2,08	2,33	0	0,39	0,65	5	0,38	0,84	5	0,39	0,65	5	0,38	0,84	5	0,26	0,65
		1	1,90	2,26	0	2,10	2,58	0	0,50	0,97	5	0,41	0,73	6	0,50	0,97	5	0,41	0,73	6	0,29	0,61
5–2	2	0,1	77,98	84,51	0	14,65	17,80	0	0,34	0,52	9	0,69	1,08	1	2,13	3,47	0	3,00	4,40	0	0,33	0,52
		1	78,48	88,90	0	14,93	20,90	0	0,41	0,69	10	0,77	1,50	0	1,99	2,66	0	2,53	4,14	0	0,41	0,69
	3	0,1	84,45	88,44	0	15,70	18,89	0	1,89	2,79	0	0,91	1,29	10	1,87	2,63	0	1,99	2,82	0	0,91	1,29
		1	83,53	88,53	0	16,25	21,18	0	1,79	2,50	0	0,73	1,19	9	1,80	2,58	1	1,74	2,52	0	0,71	1,06
	10	0,1	26,76	29,31	0	5,57	6,36	0	1,45	2,21	5	1,50	2,62	4	1,48	2,21	6	1,50	2,62	5	1,23	1,78
		1	24,49	26,91	0	5,87	6,57	0	1,81	2,39	6	1,96	2,81	4	1,81	2,39	6	1,96	2,81	4	1,61	2,35
5–5	2	0,1	48,98	50,97	0	5,81	7,79	0	1,32	2,41	0	0,59	1,33	9	1,31	2,41	1	1,31	1,62	0	0,57	1,20
		1	49,81	56,52	0	6,32	8,18	0	1,64	2,42	0	0,46	0,71	8	1,65	2,40	1	1,60	2,46	1	0,44	1,70
	3	0,1	31,42	32,40	0	5,62	6,08	0	1,51	1,87	5	1,70	2,98	3	1,52	1,99	4	1,70	2,97	1	1,27	1,79
		1	31,69	34,30	0	6,17	7,68	0	1,96	3,10	2	1,26	2,34	8	1,96	3,13	2	1,52	2,34	2	1,21	1,99
	10	0,1	5,76	6,94	0	6,11	6,82	0	1,89	2,62	2	1,69	2,50	8	1,91	2,62	2	1,70	2,50	7	1,59	2,50
		1	5,51	6,59	0	5,94	6,85	0	1,95	2,94	3	1,55	2,52	7	1,95	2,94	3	1,55	2,52	7	1,39	2,21
10–1	2	0,1	1,25	1,62	0	1,36	2,09	0	0,20	0,41	10	0,20	0,41	10	1,13	1,65	1	1,13	1,65	1	0,20	0,41
		1	1,41	1,72	0	1,55	1,92	0	0,19	0,34	10	0,19	0,34	10	0,95	1,65	0	0,95	1,65	0	0,19	0,34
	3	0,1	2,12	3,22	1	2,19	3,40	1	0,53	1,20	10	0,53	1,20	10	1,56	2,37	0	1,56	2,37	0	0,53	1,20
		1	2,43	3,06	0	2,77	3,83	0	0,49	0,62	10	0,49	0,62	10	1,63	2,47	0	1,63	2,47	0	0,49	0,62
	10	0,1	15,41	23,96	0	16,19	24,01	0	4,62	6,58	5	4,37	6,58	9	5,90	7,67	1	5,90	7,67	1	4,18	5,57
		1	26,15	29,02	0	27,59	31,74	0	6,25	7,96	7	6,80	8,20	2	6,89	7,72	3	6,97	8,06	2	6,09	7,63
10–10	2	0,1	49,12	51,56	0	12,76	14,08	0	4,49	5,97	0	1,88	2,79	10	4,40	5,72	0	3,80	4,82	0	1,88	2,79
		1	48,87	51,60	0	12,34	14,17	0	4,02	5,30	0	2,15	3,15	8	3,99	5,20	2	3,47	5,17	0	2,05	3,15
	3	0,1	29,93	31,22	0	12,14	14,84	0	3,98	6,59	3	3,40	5,00	4	3,98	6,59	3	3,78	4,76	2	3,01	4,02
		1	30,91	34,16	0	12,72	13,97	0	4,40	5,55	1	3,18	6,21	5	4,45	5,55	2	3,64	6,36	6	2,91	5,30
	10	0,1	12,17	13,15	0	11,97	12,94	0	3,72	4,42	7	4,04	4,96	3	3,72	4,42	7	4,04	4,96	3	3,49	4,29
		1	12,24	14,12	0	12,19	14,26	0	4,27	5,87	7	4,61	6,53	3	4,27	5,87	7	4,61	6,53	3	3,84	4,39

TABLE A.1 – Comparaison des heuristiques pour $HF2(m_1, Bm_2)$ lorsque $c_2 = 1$.

m_1-m_2	k	α	H_{LPT}						H_J						H_{LBPT}						H^*	
			Normal			Inverse			Normal			Inverse			Normal			Inverse				
			Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max
1–1	2	0,1	95,08	98,80	0	19,31	22,12	0	0,03	0,14	10	0,03	0,14	10	0,25	1,06	7	0,25	1,06	7	0,03	0,14
		1	93,99	98,36	0	19,32	23,35	0	0,14	0,31	10	0,14	0,31	10	0,66	1,31	3	0,66	1,31	3	0,14	0,31
	3	0,1	64,11	68,25	0	3,87	6,13	0	0,00	0,10	10	0,00	0,10	10	0,00	0,10	10	0,00	0,10	10	0,00	0,10
		1	66,45	70,72	0	4,34	5,89	0	0,00	0,02	10	0,00	0,02	10	0,00	0,02	10	0,00	0,02	10	0,00	0,02
	10	0,1	23,40	25,44	0	1,72	1,82	0	0,01	0,02	10	0,01	0,02	10	0,01	0,02	10	0,01	0,02	10	0,01	0,02
		1	20,49	21,98	0	1,69	1,79	0	0,09	0,14	10	0,09	0,14	10	0,09	0,14	10	0,09	0,14	10	0,09	0,14
1–10	2	0,1	5,54	6,91	0	1,68	1,81	0	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01
		1	5,09	5,81	0	1,66	1,73	0	0,00	0,02	10	0,00	0,02	10	0,00	0,02	10	0,00	0,02	10	0,00	0,02
	3	0,1	1,66	1,75	0	1,66	1,75	0	0,00	0,02	10	0,00	0,02	10	0,00	0,02	10	0,00	0,02	10	0,00	0,02
		1	1,70	1,81	0	1,68	1,76	0	0,01	0,06	10	0,01	0,06	10	0,01	0,06	10	0,01	0,06	10	0,01	0,06
	10	0,1	1,72	1,89	0	1,72	1,89	0	0,00	0,02	10	0,00	0,02	10	0,00	0,02	10	0,00	0,02	10	0,00	0,02
		1	1,71	1,75	0	1,71	1,75	0	0,08	0,11	10	0,08	0,11	10	0,08	0,11	10	0,08	0,11	10	0,08	0,11
2–2	2	0,1	95,66	97,88	0	20,55	25,05	0	0,38	0,64	5	0,41	1,25	4	0,98	2,32	0	1,03	2,32	1	0,22	0,48
		1	97,50	98,95	0	21,64	25,35	0	0,35	0,54	7	0,71	1,48	2	1,04	1,76	0	1,28	2,39	1	0,30	0,54
	3	0,1	68,99	70,67	0	7,07	9,81	0	0,33	0,75	3	0,21	0,35	6	0,29	0,68	2	0,40	0,71	1	0,15	0,30
		1	65,51	69,41	0	6,05	8,55	0	0,33	0,78	0	0,18	0,35	5	0,32	0,82	1	0,19	0,49	5	0,10	0,25
	10	0,1	21,51	24,67	0	3,73	3,96	0	0,29	0,57	4	0,25	0,72	4	0,32	0,53	5	0,26	0,74	4	0,18	0,47
		1	18,71	19,82	0	3,84	4,35	0	0,87	1,05	4	0,55	0,87	6	0,57	1,05	4	0,55	0,87	6	0,41	0,65
2–5	2	0,1	36,26	38,76	0	3,63	4,14	0	0,38	0,74	0	0,14	0,25	9	0,38	0,73	1	0,54	0,79	0	0,12	0,25
		1	37,86	40,34	0	3,68	3,92	0	0,37	0,66	1	0,14	0,36	5	0,37	0,64	1	0,35	0,88	4	0,10	0,36
	3	0,1	24,12	26,15	0	3,64	4,00	0	0,31	0,74	7	0,40	0,90	3	0,26	0,70	7	0,44	0,90	1	0,17	0,35
		1	24,00	26,78	0	3,77	4,20	0	0,41	0,78	1	0,25	0,51	7	0,41	0,78	2	0,36	0,81	5	0,23	0,51
	10	0,1	3,68	4,00	0	3,57	4,04	0	0,26	0,74	7	0,46	0,68	3	0,26	0,74	7	0,46	0,68	3	0,15	0,36
		1	3,75	4,13	0	3,86	4,16	0	0,61	0,94	5	0,51	0,72	6	0,61	0,94	5	0,51	0,72	6	0,44	0,66
5–2	2	0,1	40,50	43,54	0	3,28	4,70	0	0,32	0,50	9	0,85	1,71	1	1,10	1,64	0	1,80	2,45	0	0,32	0,50
		1	39,60	41,29	0	3,38	4,52	0	0,22	0,45	10	0,95	1,73	0	0,97	1,19	0	1,63	2,39	0	0,22	0,45
	3	0,1	57,43	60,57	0	7,54	9,52	0	0,75	1,40	10	1,31	2,28	0	1,82	2,09	0	2,91	4,04	0	0,75	1,40
		1	60,61	63,60	0	9,22	11,79	0	1,01	1,78	9	1,99	3,15	1	2,11	2,79	0	3,26	3,84	0	1,00	1,48
	10	0,1	57,76	65,91	0	10,36	11,14	0	1,80	2,56	3	1,63	2,34	4	1,82	2,56	3	1,68	2,66	2	1,41	2,01
		1	54,24	53,72	0	10,14	11,32	0	1,94	3,09	3	1,70	2,58	7	1,94	3,09	3	2,37	3,10	3	1,50	2,20
5–5	2	0,1	95,65	97,82	0	28,17	32,71	0	1,69	2,71	7	2,79	6,00	3	3,54	4,83	0	5,18	7,55	0	1,12	1,96
		1	92,79	95,82	0	23,47	27,61	0	1,38	1,86	7	2,92	5,18	2	3,30	5,01	1	5,26	7,21	0	1,25	1,86
	3	0,1	65,11	69,64	0	11,72	14,43	0	1,62	2,36	2	1,14	1,68	7	1,54	2,34	3	1,87	3,01	0	1,05	1,67
		1	64,61	70,50	0	12,23	14,50	0	1,84	2,36	1	0,87	1,78	1	1,85	2,34	1	1,67	2,42	1	0,85	1,63
	10	0,1	14,48	15,54	0	9,35	10,34	0	1,14	2,07	8	1,68	2,50	1	1,12	2,07	9	1,65	2,50	1	1,00	2,01
		1	12,48	14,45	0	10,13	11,16	0	2,04	2,88	5	2,01	3,18	5	2,04	2,88	5	2,01	3,18	5	1,71	2,69
10–1	2	0,1	0,63	0,87	0	0,63	0,87	0	0,09	0,22	10	0,54	0,79	0	0,54	0,79	0	0,09	0,22	0	0,09	0,22
		1	0,66	0,83	0	0,66	0,83	0	0,05	0,11	9	0,05	0,11	9	0,51	0,82	1	0,51	0,82	1	0,05	0,09
	3	0,1	1,00	1,30	0	1,01	1,30	0	0,30	0,51	10	0,30	0,51	10	0,89	1,17	0	0,89	1,17	0	0,30	0,51
		1	1,07	1,26	0	1,07	1,26	0	0,31	0,69	10	0,31	0,69	10	0,87	1,21	0	0,87	1,21	0	0,31	0,69
	10	0,1	2,53	3,56	4	2,59	3,56	4	2,03	2,92	9	2,03	2,92	9	3,30	3,76	0	3,30	3,76	0	2,00	2,92
		1	6,91	8,32	0	7,41	9,15	0	3,02	3,88	9	2,97	3,88	10	3,78	4,29	0	3,78	4,29	0	2,97	3,88
10–10	2	0,1	92,62	97,03	0	32,78	38,22	0	3,70	4,58	7	6,78	10,83	3	6,64	9,36	0	11,52	14,18	0	3,14	4,58
		1	92,48	95,67	0	32,36	34,80	0	3,63	5,20	7	6,47	10,80	2	6,23	8,78	1	10,82	15,20	0	3,36	4,60
	3	0,1	62,35	68,59	0	20,29	22,47	0	3,94	5,34	0	2,84	4,16	6	3,96	5,29	2	4,45	6,34	2	2,60	3,77
		1	62,34	68,47	0	20,88	23,55	0	4,06	5,37	0	3,17	6,37	6	4,08	5,46	2	4,51	6,81	2	2,71	3,21
	10	0,1	21,52	24,27	0	20,75	23,06	0	4,11	5,44	6	5,05	6,75	3	4,10	5,44	7	5,02	6,75	3	3,94	6,75
		1	20,55	22,91	0	20,56	23,41	0	4,84	6,68	6	5,03	6,80	4	4,84	6,68	6	5,03	6,80	4	4,27	5,80

TABLE A.2 – Comparaison des heuristiques pour $HF2(m_1, Bm_2)$ lorsque $c_2 = 2$.

m_1-m_2	k	α	H_{LPT}						H_J						H_{LBPT}						H^*	
			Normal			Inverse			Normal			Inverse			Normal			Inverse				
			Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max
1–1	2	0,1	64,85	69,17	0	8,04	10,23	0	0,09	0,14	10	0,09	0,14	10	0,58	0,75	0	0,58	0,75	0	0,09	0,75
		1	65,07	71,42	0	7,80	10,81	0	0,13	0,17	10	0,13	0,17	10	0,47	0,84	0	0,47	0,84	0	0,13	0,17
	3	0,1	93,29	97,17	0	18,81	22,45	0	0,25	0,54	10	0,25	0,54	10	0,74	2,05	4	0,74	2,05	4	0,25	0,54
		1	95,43	98,61	0	20,58	22,88	0	0,23	0,56	10	0,23	0,56	10	0,74	1,74	5	0,74	1,74	5	0,23	0,56
	10	0,1	34,52	36,71	0	2,54	2,68	0	0,01	0,02	10	0,01	0,02	10	0,01	0,02	10	0,01	0,02	10	0,01	0,02
		1	30,77	33,78	0	2,52	2,63	0	0,10	0,18	10	0,10	0,18	10	0,10	0,18	10	0,10	0,18	10	0,10	0,18
1–10	2	0,1	10,70	12,83	0	2,57	2,74	0	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01	10	0,00	0,01
		1	10,62	11,85	0	2,53	2,64	0	0,02	0,05	10	0,02	0,05	10	0,02	0,05	10	0,02	0,05	10	0,02	0,05
	3	0,1	3,55	5,52	0	2,52	2,76	0	0,00	0,02	10	0,00	0,02	10	0,00	0,02	10	0,00	0,02	10	0,00	0,02
		1	3,54	4,72	0	2,57	2,68	0	0,01	0,03	10	0,01	0,03	10	0,01	0,03	10	0,01	0,03	10	0,01	0,03
	10	0,1	2,53	2,71	0	2,53	2,71	0	0,01	0,02	10	0,01	0,02	10	0,01	0,02	10	0,01	0,02	10	0,01	0,02
		1	2,52	2,63	0	2,52	2,63	0	0,10	0,18	10	0,10	0,18	10	0,10	0,18	10	0,10	0,18	10	0,10	0,18
2–2	2	0,1	67,03	72,37	0	10,00	13,43	0	0,29	0,44	10	0,98	1,59	0	1,29	1,64	0	1,73	2,06	0	0,29	0,44
		1	65,05	70,48	0	9,20	13,86	0	0,29	0,51	8	0,66	1,42	2	0,94	1,89	0	1,51	1,99	0	0,28	0,50
	3	0,1	95,70	98,81	0	22,15	25,30	0	0,80	1,33	8	1,04	2,65	2	1,94	3,04	0	2,32	3,57	0	0,71	1,33
		1	95,84	99,14	0	23,62	26,77	0	0,67	1,12	7	1,09	2,51	2	1,47	2,69	0	1,78	3,05	1	0,60	1,12
	10	0,1	32,51	35,51	0	5,26	6,05	0	0,21	0,38	6	0,55	0,89	2	0,22	0,55	7	0,55	0,89	1	0,19	0,38
		1	28,94	30,51	0	5,39	5,78	0	0,62	0,99	5	0,57	1,13	5	0,62	0,99	5	0,57	1,13	5	0,45	0,67
2–5	2	0,1	58,63	64,80	0	6,45	7,92	0	0,44	0,92	1	0,11	0,20	8	0,44	0,92	0	0,35	0,82	2	0,09	0,15
		1	60,21	65,99	0	6,94	8,63	0	0,42	0,90	2	0,14	0,25	7	0,42	0,90	1	0,34	0,68	1	0,12	0,25
	3	0,1	37,65	41,36	0	5,36	5,88	0	0,31	0,59	2	0,23	0,46	4	0,30	0,59	4	0,31	0,75	2	0,12	0,21
		1	36,48	37,75	0	5,30	5,68	0	0,32	0,71	1	0,18	0,55	5	0,32	0,73	3	0,37	0,67	2	0,08	0,20
	10	0,1	5,36	6,25	0	5,26	6,05	0	0,22	0,55	8	0,55	0,89	2	0,22	0,55	8	0,55	0,89	2	0,21	0,55
		1	5,33	5,68	0	5,34	5,75	0	0,53	0,81	7	0,58	0,85	3	0,53	0,81	7	0,58	0,85	3	0,48	0,73
5–2	2	0,1	27,14	29,30	0	1,36	2,01	0	0,23	0,35	10	0,82	1,52	0	0,91	1,12	0	1,73	2,06	0	0,23	0,35
		1	26,18	28,30	0	1,46	2,28	0	0,32	0,92	7	0,71	1,50	3	0,63	0,95	0	1,51	1,99	0	0,24	0,46
	3	0,1	39,56	42,87	0	3,79	4,62	0	0,56	0,75	9	1,13	2,38	1	1,46	1,82	0	2,55	2,86	0	0,55	0,75
		1	39,86	42,12	0	3,73	4,67	0	0,58	0,87	10	1,45	2,67	0	1,53	1,89	0	2,51	3,05	0	0,58	0,87
	10	0,1	85,46	94,61	0	17,86	23,47	0	1,72	2,78	3	1,54	2,21	4	1,95	4,24	2	2,52	4,14	1	1,16	2,15
		1	75,53	78,43	0	18,10	23,96	0	2,43	3,32	1	2,07	3,40	4	2,45	3,25	1	1,85	2,63	5	1,54	2,44
5–5	2	0,1	65,09	71,74	0	12,44	17,94	0	0,78	1,12	10	3,54	5,74	0	2,90	3,54	0	5,53	6,25	0	0,78	1,12
		1	63,66	68,20	0	11,91	16,21	0	1,07	1,51	10	3,64	5,80	0	2,28	2,82	0	4,49	6,07	0	1,07	1,51
	3	0,1	94,02	97,29	0	27,47	32,52	0	2,17	3,35	9	4,39	6,56	1	4,66	6,88	0	7,36	10,01	0	2,09	3,35
		1	94,78	97,50	0	27,85	30,71	0	2,51	3,38	8	3,93	7,75	2	4,59	6,92	0	6,93	9,45	0	2,26	3,38
	10	0,1	25,75	28,56	0	13,77	16,98	0	1,43	3,34	6	2,10	3,17	3	1,40	3,34	6	2,07	3,17	1	1,28	2,80
		1	21,51	23,50	0	13,96	15,05	0	2,03	2,65	6	2,18	3,15	4	2,03	2,65	6	2,18	3,15	4	1,78	2,34
10–1	2	0,1	0,41	0,53	0	0,41	0,53	0	0,04	0,08	10	0,04	0,08	10	0,32	0,48	0	0,32	0,48	0	0,04	0,08
		1	0,41	0,54	0	0,41	0,54	0	0,06	0,12	10	0,06	0,12	10	0,32	0,53	0	0,32	0,53	0	0,06	0,12
	3	0,1	0,57	0,80	0	0,57	0,80	0	0,20	0,38	10	0,20	0,38	10	0,56	0,76	1	0,56	0,76	1	0,20	0,38
		1	0,53	0,86	1	0,53	0,86	1	0,17	0,24	10	0,17	0,24	10	0,69	0,86	0	0,69	0,86	0	0,17	0,24
	10	0,1	1,95	2,43	6	1,97	2,43	5	1,65	2,20	8	1,65	2,20	8	2,31	2,47	1	2,31	2,47	1	1,63	2,20
		1	3,48	4,10	0	3,57	4,53	0	2,08	2,59	9	2,08	2,59	9	2,52	2,64	2	2,52	2,64	2	2,08	2,57
10–10	2	0,1	64,44	69,03	0	17,27	19,29	0	2,39	2,65	10	8,21	11,67	0	5,82	7,41	0	11,80	14,64	0	2,39	2,65
		1	63,89	77,47	0	16,75	26,60	0	2,53	3,80	10	8,85	11,03	0	5,47	6,95	0	11,62	13,82	0	2,53	3,80
	3	0,1	89,97	95,19	0	36,25	41,15	0	5,88	8,02	10	10,77	15,66	0	8,98	11,75	0	14,42	18,09	0	5,88	8,02
		1	91,97	95,82	0	36,92	43,37	0	6,40	7,68	10	13,00	19,24	0	10,85	14,06	0	16,65	19,66	0	6,40	7,68
	10	0,1	27,85	30,34	0	28,44	30,54	0	4,16	5,68	5	3,90	5,12	4	4,15	5,68	5	3,88	5,12	5	3,14	4,11
		1	28,40	31,22	0	28,52	30,80	0	4,88	6,42	6	4,97	6,81	4	4,88	6,42	6	4,97	6,80	4	4,46	6,34

TABLE A.3 – Comparaison des heuristiques pour $HF2(m_1, Bm_2)$ lorsque $c_2 = 3$.

m_1-m_2	k	α	H_{LPT}						H_J						H_{LBPT}						H^*	
			Normal			Inverse			Normal			Inverse			Normal			Inverse				
			Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max	Best	Moy	Max
1–1	2	0,1	50,19	53,56	0	3,46	4,77	0	0,06	0,11	10	0,06	0,11	10	0,43	0,65	0	0,43	0,65	0	0,06	0,11
		1	48,24	56,93	0	3,93	6,32	0	0,09	0,14	10	0,09	0,14	10	0,38	0,55	0	0,38	0,55	0	0,09	0,14
	3	0,1	73,08	79,80	0	11,87	14,43	0	0,26	0,56	10	0,26	0,56	10	0,92	1,28	0	0,92	1,28	0	0,26	0,56
		1	74,98	78,81	0	12,31	16,35	0	0,44	0,67	10	0,44	0,67	10	1,08	1,46	0	1,08	1,46	0	0,44	0,67
	10	0,1	45,90	50,32	0	3,38	3,60	0	0,01	0,04	10	0,01	0,04	10	0,02	0,06	9	0,02	0,06	9	0,01	0,04
		1	40,72	44,19	0	3,38	3,60	0	0,13	0,22	10	0,13	0,22	10	0,13	0,22	10	0,13	0,22	10	0,13	0,22
1–10	2	0,1	15,97	17,73	0	3,41	3,69	0	0,01	0,04	10	0,01	0,04	10	0,01	0,04	10	0,01	0,04	10	0,01	0,04
		1	16,05	17,80	0	3,41	3,60	0	0,03	0,08	10	0,03	0,08	10	0,03	0,08	10	0,03	0,08	10	0,03	0,08
	3	0,1	6,35	7,53	0	3,34	3,47	0	0,00	0,02	10	0,00	0,02	10	0,00	0,02	10	0,00	0,02	10	0,00	0,02
		1	6,57	9,19	0	3,39	3,68	0	0,01	0,03	10	0,01	0,03	10	0,01	0,03	10	0,01	0,03	10	0,01	0,03
	10	0,1	3,38	3,57	0	3,38	3,57	0	0,01	0,03	10	0,01	0,03	10	0,01	0,03	10	0,01	0,03	10	0,01	0,03
		1	3,39	3,61	0	3,39	3,61	0	0,14	0,20	10	0,14	0,20	10	0,14	0,20	10	0,14	0,20	10	0,14	0,20
2–2	2	0,1	50,19	53,85	0	4,47	7,18	0	0,22	0,32	7	0,41	1,18	4	0,86	1,23	0	1,50	1,81	0	0,19	0,32
		1	49,33	55,53	0	4,86	6,77	0	0,28	0,40	8	0,69	1,20	2	0,83	1,18	0	1,47	1,78	0	0,26	0,38
	3	0,1	72,35	75,81	0	12,02	14,73	0	0,71	1,01	7	1,07	2,56	2	1,92	2,93	1	2,65	3,66	0	0,66	1,01
		1	75,47	77,72	0	13,89	16,24	0	0,89	1,48	7	1,39	2,82	3	1,85	2,19	0	2,75	3,26	0	0,84	1,29
	10	0,1	43,84	45,62	0	7,01	7,79	0	0,39	0,85	1	0,33	0,80	3	0,34	0,78	5	0,26	0,59	4	0,20	0,53
		1	38,81	42,60	0	6,89	7,64	0	0,66	1,36	6	0,75	1,03	3	0,64	1,11	6	0,74	1,03	4	0,53	0,91
2–5	2	0,1	79,22	86,07	0	16,05	21,93	0	0,36	0,72	2	0,12	0,21	7	0,34	0,75	2	0,44	0,79	0	0,10	0,21
		1	79,15	84,92	0	14,53	20,70	0	0,21	0,53	2	0,10	0,19	9	0,21	0,53	1	0,46	0,81	0	0,10	0,19
	3	0,1	51,37	57,12	0	7,21	7,68	0	0,47	0,77	1	0,24	0,53	6	0,45	0,75	3	0,32	0,68	2	0,20	0,37
		1	51,19	55,69	0	7,20	8,75	0	0,36	0,56	2	0,20	0,42	5	0,36	0,55	3	0,41	0,86	3	0,15	0,35
	10	0,1	8,49	10,04	0	6,99	7,58	0	0,32	0,87	7	0,46	0,83	2	0,32	0,83	7	0,46	0,83	3	0,27	0,46
		1	7,24	7,89	0	6,93	7,36	0	0,62	1,06	6	0,70	1,00	4	0,62	1,06	6	0,70	1,00	4	0,54	0,83
5–2	2	0,1	20,27	21,90	0	0,84	1,11	0	0,36	1,09	6	0,43	1,28	3	0,61	0,87	1	1,50	1,81	0	0,22	0,57
		1	19,57	20,76	0	0,85	1,15	0	0,36	0,73	8	0,83	1,42	2	0,62	0,75	0	1,43	1,60	0	0,30	0,73
	3	0,1	29,24	30,78	0	1,75	2,62	0	0,63	1,21	6	1,07	2,56	3	1,11	1,72	1	2,30	2,68	0	0,55	1,06
		1	30,55	31,45	0	2,01	2,76	0	0,74	1,87	8	1,05	2,45	2	1,21	1,45	0	2,46	2,73	0	0,58	0,84
	10	0,1	83,84	86,99	0	15,13	17,75	0	3,38	4,47	7	3,38	4,70	4	8,10	8,95	0	9,78	10,84	0	3,08	4,11
		1	93,20	97,35	0	28,35	32,06	0	5,93	8,51	5	7,08	9,93	2	6,97	8,95	3	8,38	10,37	0	5,76	7,68
5–5	2	0,1	49,25	52,33	0	6,60	8,00	0	0,88	1,10	10	3,09	5,42	0	2,42	2,94	0	5,07	5,72	0	0,88	1,10
		1	48,29	54,54	0	6,75	7,95	0	0,75	1,18	10	3,05	4,83	0	2,10	2,58	0	4,78	5,67	0	0,75	1,18
	3	0,1	72,25	77,21	0	16,26	19,77	0	2,39	3,08	10	4,97	8,17	0	4,55	5,44	0	7,90	8,91	0	2,39	3,08
		1	72,51	80,00	0	16,66	20,35	0	2,39	3,05	10	6,28	7,72	0	4,93	5,27	0	8,26	9,71	0	2,39	3,05
	10	0,1	36,40	38,96	0	17,95	19,23	0	1,55	2,40	4	1,80	3,16	5	1,56	2,50	3	1,98	3,06	2	1,32	2,04
		1	31,80	34,87	0	17,55	18,86	0	2,10	3,24	6	2,22	3,65	3	2,07	3,03	7	2,18	3,16	3	1,74	2,95
10–1	2	0,1	0,29	0,40	0	0,29	0,40	0	0,04	0,11	10	0,04	0,11	10	0,25	0,40	0	0,25	0,40	0	0,04	0,11
		1	0,29	0,41	0	0,29	0,41	0	0,04	0,06	10	0,04	0,06	10	0,28	0,43	0	0,28	0,43	0	0,04	0,06
	3	0,1	0,44	0,64	2	0,44	0,64	2	0,17	0,24	9	0,17	0,24	9	0,42	0,63	1	0,42	0,63	1	0,16	0,24
		1	0,43	0,61	0	0,43	0,61	0	0,12	0,19	10	0,12	0,19	10	0,43	0,63	0	0,43	0,63	0	0,12	0,19
	10	0,1	1,38	1,89	2	1,38	1,89	2	0,86	1,40	9	0,86	1,40	9	1,66	1,90	0	1,66	1,90	0	0,83	1,40
		1	2,23	2,76	0	2,51	3,18	0	1,77	2,08	7	1,77	2,08	7	1,92	1,20	3	1,92	2,20	3	1,68	1,98
10–10	2	0,1	47,55	51,03	0	9,89	11,83	0	2,71	6,66	9	9,16	11,65	0	4,34	5,30	1	11,14	12,49	0	2,51	4,61
		1	47,46	55,82	0	10,17	13,88	0	3,54	5,59	7	9,20	11,89	0	4,00	5,01	3	9,75	11,21	0	3,38	5,01
	3	0,1	72,08	78,53	0	24,49	28,48	0	5,49	6,60	10	12,10	16,85	0	9,25	10,63	0	16,71	18,72	0	5,49	6,60
		1	72,94	78,57	0	24,75	29,14	0	6,09	7,19	10	13,54	15,31	0	9,48	11,16	0	16,67	19,44	0	6,09	7,19
	10	0,1	36,35	38,33	0	36,58	39,73	0	4,41	6,30	4	4,45	6,16	3	4,41	6,30	5	4,41	6,07	5	3,81	5,42
		1	36,36	39,05	0	37,12	39,97	0	5,48	7,18	3	4,89	5,96	7	5,48	7,18	3	4,89	5,96	7	4,76	5,96

TABLE A.4 – Comparaison des heuristiques pour $HF2(m_1, Bm_2)$ lorsque $c_2 = 4$.

Annexe B

Résultats expérimentaux des méthodes exactes pour des instances de 15 tâches.

n	k	α	c_2	Heuristiques		Directe (PSE)						Inverse (dynamique)						Inverse (PSE)					
				Écart		Temps		Non	Non	Écart		Temps		Non	Non	Écart		Temps		Non	Non	Écart	
				Moy	Max	Moy	Max	Opt.	Best	Moy	Max	Moy	Max	Opt.	Best	Moy	Max	Moy	Max	Opt.	Best	Moy	Max
2-2	2	0,1	1	1,8	6,9	1	2	8	0	0,6	1,3	0	0	6	2	0,5	1,0	0	0	6	2	0,6	1,3
			2	3,5	6,1			10	7	4,2	9,5	13	59	0	0	0,0	0,0	65	364	0	0	0,0	0,0
			3	3,5	9,8			10	10	3,7	8,3	14	61	0	0	0,0	0,0	18	62	0	0	0,0	0,0
			4	4,2	7,3			10	9	4,3	8,1	60	334	1	0	0,6	0,6	68	325	1	0	0,6	0,6
		1	1	3,6	3,2			10	2	3,2	1,1	0	0	7	4	1,9	4,7	0	0	7	4	2,1	4,7
			2	4,7	8,7			10	10	6,7	11,7	166	405	1	0	6,1	6,1	89	147	7	4	4,8	6,5
			3	4,9	10,9			10	10	7,4	8,5	208	485	1	0	4,5	4,5	194	357	6	1	4,5	5,7
			4	6,3	10,7			10	10	5,5	9,0	175	525	1	0	4,3	4,3	275	568	4	1	3,3	5,5
	3	0,1	1	2,5	5,4			9	0	1,0	2,3	119	396	4	0	0,6	1,0	124	495	6	2	1,3	3,8
			2	4,3	6,7			10	5	4,4	8,5	1	7	0	0	0,0	0,0	8	69	0	0	0,0	0,0
			3	5,4	10,9			10	8	6,4	11,4	3	12	0	0	0,0	0,0	8	22	0	0	0,0	0,0
			4	5,2	12,9			10	7	3,9	6,2	9	24	0	0	0,0	0,0	37	204	0	0	0,0	0,0
		1	1	6,1	11,6			10	2	1,1	3,4	188	467	6	2	1,4	5,4	3	5	8	2	1,5	5,4
			2	6,5	12,3	0	0	9	3	3,1	8,7	2	5	1	1	6,5	6,5	233	555	2	2	5,3	6,5
			3	6,4	13,2			10	10	19,4	27,9	22	61	0	0	0,0	0,0	162	231	7	7	18,0	23,0
			4	7,6	17,1			10	10	18,6	25,6	28	87	0	0	0,0	0,0	150	150	9	4	14,2	20,0
5-5	2	0,1	1	9,4	12,4	175	560	0	0	0,0	0,0	249	517	2	2	9,5	12,4	234	297	8	7	9,6	16,9
			2	10,7	19,5			10	7	13,3	26,0	2	6	0	0	0,0	0,0	4	14	0	0	0,0	0,0
			3	9,2	18,7	0	0	9	9	16,5	30,7	1	2	0	0	0,0	0,0	0	1	0	0	0,0	0,0
			4	11,3	19,6			10	10	13,2	20,3	1	3	0	0	0,0	0,0	1	3	0	0	0,0	0,0
		1	1	12,0	18,6	156	372	3	0	3,9	6,0	142	253	7	6	7,1	10,2	599	599	9	8	8,1	14,5
			2	12,2	20,3			10	10	15,2	35,1	51	144	0	0	0,0	0,0	61	120	2	1	5,7	8,6
			3	12,6	23,3	1	2	8	8	16,0	34,8	30	105	1	1	4,8	4,8	35	101	0	0	0,0	0,0
			4	11,9	16,4	0	0	9	9	18,6	27,1	37	159	0	0	0,0	0,0	29	115	0	0	0,0	0,0
	3	0,1	1	14,0	23,2	209	563	2	1	7,3	7,9	104	577	1	1	10,2	10,2			7	6	9,7	17,7
			2	10,6	18,6			10	10	15,5	27,3	1	7	0	0	0,0	0,0	1	5	0	0	0,0	0,0
			3	11,3	22,0			9	9	11,2	27,2	7	66	0	0	0,0	0,0	1	4	0	0	0,0	0,0
			4	9,7	17,8	0	0	9	9	13,1	24,3	0	0	0	0	0,0	0,0	0	0	0	0	0,0	0,0
		1	1	16,5	22,8	86	270	3	1	9,2	11,9	122	347	3	1	8,3	12,2	599	599	9	9	10,5	16,5
			2	7,1	11,0	5	19	6	5	7,5	18,2	27	100	0	0	0,0	0,0	128	443	2	2	8,4	13,4
			3	4,7	14,3	60	208	4	4	7,2	9,0	10	35	0	0	0,0	0,0	44	227	4	2	4,2	7,6
			4	2,9	8,2	37	101	4	4	3,5	5,9	13	35	0	0	0,0	0,0	203	597	0	0	0,0	0,0

TABLE B.1 – Comparaison des PSE pour m_1-m_2 =2-2 et 5-5 lorsqu'il y a 15 tâches.

n	k	α	c_2	Heuristiques		Directe (PSE)						Inverse (dynamique)						Inverse (PSE)					
				Écart		Temps		Non	Non	Écart		Temps		Non	Non	Écart		Temps		Non	Non	Écart	
				Moy	Max	Moy	Max	Opt.	Best	Moy	Max	Moy	Max	Opt.	Best	Moy	Max	Moy	Max	Opt.	Best	Moy	Max
2-5	2	0,1	1	2,9	6,8			10	0	0,5	0,8	0	0	9	4	1,0	2,1	0	0	9	4	1,0	2,1
			2	2,1	5,0			10	0	1,8	4,3	62	165	5	0	2,3	3,6	11	15	8	2	1,9	3,6
			3	5,0	10,2			10	6	4,5	11,7	5	22	0	0	0,0	0,0	127	249	3	2	4,5	9,3
			4	10,2	18,6			10	8	17,4	26,4	2	12	0	0	0,0	0,0	98	468	2	1	8,4	11,7
		1	1	3,3	5,7	1	2	8	0	0,7	1,7	2	3	4	2	1,9	3,7	2	3	4	2	1,9	3,7
			2	3,2	7,2			10	0	1,6	4,3	40	185	5	1	2,7	6,8	15	50	6	2	2,8	6,8
			3	5,4	10,7			10	5	5,1	13,7	179	466	2	1	3,9	6,5	5	5	9	8	7,6	12,7
			4	9,1	19,0			10	9	12,6	22,5	74	109	0	0	0,0	0,0			10	9	11,3	18,4
	3	0,1	1	3,7	7,7			10	0	0,6	1,3	42	254	4	3	1,9	3,4	31	188	4	3	1,9	3,4
			2	4,2	7,7			10	0	2,3	6,0	81	428	1	1	4,6	4,6	8	17	8	2	2,8	6,0
			3	3,5	7,7			10	3	3,8	11,9	4	20	1	0	0,8	0,8	176	336	6	3	5,5	14,6
			4	6,9	15,5			10	7	11,0	30,0	2	8	0	0	0,0	0,0	122	314	4	2	2,0	3,0
		1	1	4,9	9,7			10	0	0,5	1,4	64	194	3	0	0,4	0,7	11	35	4	0	0,4	0,7
			2	5,4	11,1			10	1	2,4	5,1	151	420	0	0	0,0	0,0	118	348	3	1	1,6	2,6
			3	7,9	14,6			10	4	6,5	23,1	67	160	1	0	1,6	1,6			10	8	9,3	22,5
			4	7,4	11,3			10	10	7,1	19,2	39	103	0	0	0,0	0,0			10	10	9,6	17,8
5-2	2	0,1	1	4,8	8,9			10	9	6,1	18,6	22	65	2	1	1,1	1,6	19	61	2	1	1,1	1,6
			2	5,45	9,27			10	10	4,39	7,95	93	266	2	0	0,5	0,6	86	267	2	0	0,5	0,6
			3	3,6	9,3			10	10	2,4	4,5	132	468	2	0	0,5	0,8	132	468	2	0	0,5	0,8
			4	4,7	7,4			10	9	2,4	4,1	115	327	1	0	0,2	0,2	114	324	1	0	0,2	0,2
		1	1	8,0	16,8			10	10	9,0	13,0			10	0	5,2	9,7			10	0	5,2	9,7
			2	6,5	10,1			10	8	5,8	9,9	397	574	7	0	3,8	4,8	392	575	7	0	3,8	4,8
			3	4,6	9,0			10	7	3,4	6,1	194	343	3	0	2,8	3,2	193	343	3	0	2,8	3,2
			4	4,9	7,3			10	10	4,6	7,0	210	452	1	0	1,9	1,9	209	450	1	0	1,9	1,9
	3	0,1	1	5,9	10,4			9	8	7,8	13,0	19	129	0	0	0,0	0,0	19	129	0	0	0,0	0,0
			2	2,3	5,9			10	6	3,9	9,0	14	54	0	0	0,0	0,0	14	54	0	0	0,0	0,0
			3	6,3	10,4			10	6	3,0	7,0	6	19	0	0	0,0	0,0	6	19	0	0	0,0	0,0
			4	5,3	9,3			10	6	3,4	5,4	11	61	0	0	0,0	0,0	11	61	0	0	0,0	0,0
		1	1	10,4	20,7			10	9	18,6	31,4	11	27	0	0	0,0	0,0	18	41	0	0	0,0	0,0
			2	7,6	17,4			10	10	15,3	22,7	66	361	0	0	0,0	0,0	64	358	0	0	0,0	0,0
			3	9,7	17,7			10	10	13,2	17,3	11	19	0	0	0,0	0,0	10	15	0	0	0,0	0,0
			4	7,7	11,0			10	9	10,3	13,7	11	55	0	0	0,0	0,0	11	55	0	0	0,0	0,0

TABLE B.2 – Comparaison des PSE pour m_1-m_2 =2-5 et 5-2 lorsqu'il y a 15 tâches.

Bibliographie

[1] Abiri, M. B., M. Zandieh, and A. Alem-Tabriz. (2009). A tabu search approach to hybrid flow shops scheduling with sequence-dependent setup times. *Journal of Applied Sciences*, 9(9) : 1740-1745,

[2] Afrati, F., E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. (1999). Approximation schemes for minimizing average weighted completion time with release dates. *Proceeding of the 40th annual IEEE symposium on foundations of computer science*, 32–43.

[3] Ahmadi, J.H., R.H. Ahmadi, S.Dasu, and C.S. Tang. (1992). Batching and scheduling jobs on batch and discrete processors. *Operations Research*, 40 : 750-763.

[4] Al-Anzi, F.S., and A. Allahverdi. (2006). A hybrid tabu search heuristic for the two-stage assembly scheduling problem. *International Journal of Operations Research*, 3 : 109–119.

[5] Al-Anzi, F. S., and A. Allahverdi. (2007). A self-adaptive differential evolution heuristic for two-stage assembly scheduling problem to minimize maximum lateness with setup times. *European Journal of Operational Research*, 182(1) : 80-94.

[6] Al-Anzi, F. S., and A. Allahverdi. (2009). Heuristics for a two-stage assembly flowshop with bicriteria of maximum lateness and makespan. *Computers & Operations Research*, 36(9) : 2682-2689.

[7] Alaykýran, K., O. Engin, and A. Döyen. (2007). Using ant colony optimization to solve hybrid flow shop scheduling problems. *International Journal of Advanced Manufacturing Technology*, 35(5-6) : 541-550.

[8] Alem Tabriz, A., M. Zandieh, and Z. Vaziri. (2009). A novel simulated annealing algorithm to hybrid flow shops scheduling with sequence-dependent setup times. *Journal of Applied Sciences*, 9(10) : 1943-1949.

[9] Allahverdi, A., and F. S. Al-Anzi. (2006). A PSO and a tabu search heuristics for the assembly scheduling problem of the two-stage distributed database application. *Computers & Operations Research*, 33(4) : 1056-1080.

[10] Allahverdi, A., and F. S. Al-Anzi. (2006). Evolutionary heuristics and an algorithm for the two-stage assembly scheduling problem to minimize makespan with setup times. *International Journal of Production Research*, 44(22) : 4713-4735.

[11] Allahverdi, A., and F. S. Al-Anzi. (2009). The two-stage assembly scheduling problem to minimize total completion time with setup times. *Computers & Operations Research*, 36(10) : 2740-2747.

[12] Amin-Naseri, M. R., and M. A. Beheshti-Nia. (2009). Hybrid flow shop scheduling with parallel batching. *International Journal of Production Economics*, 117(1) : 185-196.

[13] Anthony, R.N. (1965). Planning and Control Systems : a framework for analysis. *Boston, Harvard University Press.*

[14] Azizoglu, M., E. Cakmak, and S. Kondakci. (2001). A flexible flowshop problem with total flow time minimization. *European Journal of Operational Research*, 132(3) : 528-538.

[15] Azizoglu, M., and S. Webster. (2000). Scheduling a batch processing machine with non-identical job sizes. *International Journal of Production Research*, 38(10) : 2173-2184.

[16] Azizoglu, M., and S. Webster. (2001). Scheduling a batch processing machine with incompatible job families. *Computers and Industrial Engineering*, 39 : 325–335.

[17] Bai, S., Zhang, F., Li, S., and Liu, Q. (2007). Scheduling an unbounded batch machine to minimize maximum lateness. *Lecture Notes in Computer Science*, 4613 LNCS : 172-177.

[18] Balasubramanian, H., L. Mönch, J. Fowler, and M. Pfund. (2004). Genetic algorithm based scheduling of parallel batch machines with incompatible job families to minimize total weighted tardiness. *International Journal of Production Research*, 42(8) : 1621-1638.

[19] Bellanger, A., and A. Oulamara. (2009). Scheduling hybrid flowshop with parallel batching machines and compatibilities. *Computers & Operations Research*, 36 : 1982-1992

[20] Blazewicz, J., K. Ecker, G. Schmidt, and J. Werglarz. (1994). Scheduling in Computer and Manufacturing Systems. *Springer-Verlag edition.*

[21] Blazewicz, J., J. K. Lenstra, and A. H. G. Rinnooy Kan. (1983). Scheduling subject to resource constraints : classification and complexity. *Discrete Applied Mathematics*, 5(1) : 11-24.

[22] Boudhar, M., and G. Finke. (2000). Scheduling on a batch machine with job compatibilities. *Belgian Journal of Operations Research, Statistics and Computer Science*, 40(1-2) : 69-80.

[23] Boudhar, M. (2003). Dynamic scheduling on a single batch processing machine with split compatibility graphs. *Journal of Mathematical Modelling and Algorithms*, 2 : 17-35.

[24] Boudhar, M. (2003). Scheduling a batch processing machine with bipartite compatibility graphs. *Mathematical Methods of Operations Research*, 57(3) : 513-527.

[25] Boudhar, M. (2005). Scheduling on a batch processing machine with split compatibility graphs. *Journal of Mathematical Modelling and Algorithms*, 4(4) : 391-407.

[26] Brah, S. A., and J. L. Hunsucker. (1991). Branch and bound algorithm for the flow shop with multiple processors. *European Journal of Operational Research*, 51(1) : 88-99.

[27] Brah, S. A., and L. L. Loo. (1999). Heuristics for scheduling in a flow shop with multiple processors. *European Journal of Operational Research*, 113(1) : 113-122.

[28] Brucker, P., A. Gladky, H. Hoogeveen, M. Y. Kovalyov, C. N. Potts, T. Tautenhahn, and S. L. Van De Velde. (1998). Scheduling a batching machine. *Journal of Scheduling*, 1(1) : 31-54.

[29] Cai, M., X. Deng, H. Feng, G. Li, and G. Liu. (2002). A PTAS for minimizing total completion time of bounded batch scheduling. *Lecture Notes in Computer Science*, 2337 (IPCOrsquo2002), MIT : 304–314.

[30] Carlier, J. (1984). Problèmes d'Ordonnancement à Contraintes de Ressources : Algorithmes et Complexité. *Thèse d'État, MASI.*

[31] Carlier, J., and E. Néron. (2000). An exact method for solving the multiprocessor flowshop. *RAIRO-Oper Res.* 78 : 146-161.

[32] Castro, P. M., A.Q. Novais, and A. Carvalho. (2008). MILP-based decomposition method for the optimal scheduling of an industrial batch plant. *Computer Aided Chemical Engineering*, 25 : 557-562.

[33] Chandru, V., C. -Y Lee and R. Uzsoy. (1993). Minimizing total completion time on batch processing machines. *International Journal of Production Research*, 31(9) : 2097-2121.

[34] Chandru, V., C. -Y Lee, and R. Uzsoy. (1993). Minimizing total completion time on a batch processing machine with job families. *Operations Research Letters*, 13(2) : 61-65.

[35] Chang, P. -C, Y. -S Chen, and H. -M Wang. (2005). Dynamic scheduling problem of batch processing machine in semiconductor burn-in operations. *Lecture Notes in Computer Science*, 3483(IV) : 172-181.

[36] Chang, P. -Y, P. Damodaran, and S. Melouk. (2004). Minimizing makespan on parallel batch processing machines. *International Journal of Production Research*, 42(19) : 4211-4220.

[37] Chang, P.C., and H.M. Wang. (2004). A heuristic for a batch processing machine scheduled to minimize total completion time with non-identical job sizes. *International Journal of Advanced Manufacturing Technology*, 24(7-8) : 615-620.

[38] Chen, B., X. Deng, and W. Zang. (2004). On-line scheduling a batch processing system to minimize total weighted job completion time. *Journal of Combinatorial Optimization*, 8(1) : 85-95.

[39] Cheng, B. -Y, H. -P Chen, H. Shao, R. Xu, and G. Q. Huang. (2008). A chaotic ant colony optimization method for scheduling a single batch-processing machine with non-identical job sizes. *IEEE Congress on Evolutionary Computation*, CEC 2008 : 40-43.

[40] Cheng, B. -Y, H. -P Chen, and S. -S Wang. (2008). Fuzzy scheduling for single batch-processing machine with non-identical job sizes. *IEEE International Conference on Fuzzy Systems* : 27-30.

[41] Cheng, T. C. E., Z. Liu, and W. Yu. (2001). Scheduling jobs with release dates and deadlines on a batch processing machine. *IIE Transactions*, 33(8) : 685-690.

[42] Cheng, T. C. E., C. T. Ng, J. J. Yuan, and Z. H. Liu. (2004). Single machine parallel batch scheduling subject to precedence constraints. *Naval Research Logistics*, 51(7) : 949-958.

[43] Cheng, T. C. E., and G. Wang. (1999). Scheduling the fabrication and assembly of components in a two-machine flowshop. *IIE Transactions*, 31(2) : 135-143.

[44] Cheng, T. C. E., J. J. Yuan, and A. F. Yang. (2005). Scheduling a batch-processing machine subject to precedence constraints, release dates and identical processing times. *Computers & Operations Research*, 32(4) : 849-859.

[45] Cheraghi, S. H., V. Vishwaram, and K. K. Krishnan. (2003). Scheduling a single batch-processing machine with disagreeable ready times and due dates. *International Journal of Industrial Engineering : Theory Applications and Practice*, 10(2) : 175-187.

[46] Choi, H. -S, and D. -H Lee. (2009). Scheduling algorithms to minimize the number of tardy jobs in two-stage hybrid flow shops. *Computers and Industrial Engineering*, 56(1) : 113-120.

[47] Chou, F. -D. (2007). A joint GA+DP approach for single burn-in oven scheduling problems with makespan criterion. *International Journal of Advanced Manufacturing Technology*, 35(5-6) : 587-595.

[48] Chou, F. -D, P. -C Chang, and H. -M Wang. (2006). A hybrid genetic algorithm to minimize makespan for the single batch machine dynamic scheduling problem. *International Journal of Advanced Manufacturing Technology*, 31(3-4) : 350-359.

[49] Chou, F. -D, and H. -M Wang. (2008). Scheduling for a single semiconductor batch-processing machine to minimize total weighted tardiness. *Journal of the Chinese Institute of Industrial Engineers*, 25(2) : 136-147.

[50] Conway, R., W. Maxwell, and L. Miller. (1967). *Theory of scheduling.* Addison Wesley, Massachussets.

[51] Damodaran, P., and P. -Y Chang. (2008). Heuristics to minimize makespan of parallel batch processing machines. *International Journal of Advanced Manufacturing Technology*, 37(9-10) : 1005-1013.

[52] Damodaran, P., P. Kumar Manjeshwar, and K. Srihari. (2006). Minimizing makespan on a batch-processing machine with non-identical job sizes using genetic algorithms. *International Journal of Production Economics*, 103(2) : 882-891,

[53] Damodaran, P., and K. Srihari. (2004). Mixed integer formulation to minimize makespan in a flow shop with batch processing machines. *Mathematical and Computer Modelling*, 40(13) : 1465-1472.

[54] Damodaran, P., K. Srihari, and S. S. Lam. (2007). Scheduling a capacitated batch-processing machine to minimize makespan. *Robotics and Computer-Integrated Manufacturing*, 23(2) : 208-216.

[55] Danneberg, D., T. Tautenhahn, and F. Werner. (1999). A comparison of heuristic algorithms for flow shop scheduling problems with setup times and limited batch size. *Mathematical and Computer Modelling*, 29(9) : 101-126.

[56] Demange, M., D. De Werra, J. Monnot, and V. Th Paschos. (2002). Weighted node coloring : When stable sets are expensive. *Proceedings of the 28th International Workshop on Graph-theoretic Concepts in Computer Science*, 2573 : 114-125.

[57] Deng, X., H. Feng, G. Li, and B. Shi. (2005). A PTAS for semiconductor burn-in scheduling. *Journal of Combinatorial Optimization*, 9(1) : 5-17.

[58] Deng, X., H. Feng, P. Zhang, Y. Zhang, and H. Zhu. (2004). Minimizing mean completion time in a batch processing system. *Algorithmica*, 38(4) : 513-528.

[59] Deng X., C. Poon, and Y. Zhang. (2003). Approximation algorithms in batch processing. *Journal of Combinatorial Optimization*, 7 :247–257.

[60] Deng, X., and Y. Zhang. (1999) Minimizing mean response time in batch processing system. *Lecture Notes in Computer Science*, 1627 : 231-240.

[61] Devpura, A., Fowler, J.W., Carlyle, M., Perez, I. (2000) Minimizing total weighted tardiness on single batch process machine with incompatible job families. *Proceedings of the Symposium on Operations Research, Dresden, Germany* : 366-371.

[62] De Werra, D., M. Demange, B. Escoffier, J. Monnot, and V. Th Paschos. (2009). Weighted coloring on planar, bipartite and split graphs : Complexity and approximation. *Discrete Applied Mathematics*, 157(4) : 819-832.

[63] Dobson, G., and R.S. Nambinadom. (1992). The batch loading and scheduling problem. *Research Report, Simon School of Business Administration, University of Rochester N.Y.*.

[64] Dobson, G., and R.S. Nambimadom. (2001). The batch loading and scheduling problem. *Operations research*, 49(1) : 52-65.

[65] Dupont, L., and C. Dhaenens-Flipo. (2002). Minimizing the makespan on a batch machine with non-identical job sizes : An exact procedure. *Computers & Operations Research*, 29(7) : 807-819.

[66] Dupont, L. and F. Jolai Ghazvini. (1997). Branch and bound method for minimizing mean flow time on a single batch processing machine. *International Journal of Industrial Engineering : Practice and Theory*, 4 : 197-203.

[67] Dupont, L., and F. Jolai Ghazvini. (1998). Minimizing makespan on a single batch processing machine with non-identical job sizes. *Journal Européen des Systèmes Automatisés*, 32(4) : 431-440.

[68] Engin, O., and A. Döyen. (2004). A new approach to solve hybrid flow shop scheduling problems by artificial immune system. *Future Generation Computer Systems*, 20(6) : 1083-1095.

[69] Epstein, L., M. M. Halldórsson, A. Levin, and H. Shachnai. (2007). Weighted sum coloring in batch scheduling of conflicting jobs. *Algorithmica* : 1-23.

[70] Epstein, L., and Levin, A. (2008). On the max coloring problem. *Lecture Notes in Computer Science*, 4927 LNCS : 142-155.

[71] Erramilli, V., and S. J. Mason. (2006). Multiple orders per job compatible batch scheduling. *IEEE Transactions on Electronics Packaging Manufacturing*, 29(4) : 285-296.

[72] Erramilli, V., and S. J. Mason. (2008). Multiple orders per job batch scheduling with incompatible jobs. *Annals of Operations Research*, 159(1) : 245-260.

[73] Escoffier, B., J. Monnot, and V. T. Paschos. (2006). Weighted coloring : Further complexity and approximability results. *Information Processing Letters*, 97(3) : 98-103.

[74] Finke, G., V. Jost, M. Queyranne and A. Sebö. (2008). Batch processing with interval graph compatibilities between tasks. *Discrete Applied Mathematics*, 156 : 556-568.

[75] Fowler, J. W., G. L. Hogg, and D. T. Phillips. (1992). Control of multiproduct bulk service diffusion/oxidation processes. *IIE Transactions*, 24(4) : 84-96.

[76] Fowler, J. W., G. L. Hogg, and D. T. Phillips. (2000). Control of multiproduct bulk server diffusion/oxidation processes. part 2 : Multiple servers. *IIE Transactions*, 32(2) : 167-176.

[77] Fu, R., T. Ji, J. Yuan, and Y. Lin. (2007). Online scheduling in a parallel batch processing system to minimize makespan using restarts. *Theoretical Computer Science*, 374(1-3) : 196-202.

[78] Fu, R., Ji Tian, and J. Yuan. (2009). On-line scheduling on an unbounded parallel batch machine to minimize makespan of two families of jobs. *Journal of Scheduling*, 12(1) : 91-97.

[79] Fu, R., J. Tian, J. Yuan, and C. He. (2008). On-line scheduling on a batch machine to minimize makespan with limited restarts. *Operations Research Letters*, 36(2) : 255-258.

[80] Gantt, H.L. (1903). A graphical daily balance in manufacture. *ASME Transactions*, 24 : 317-322.

[81] Gao, C., and L. Tang. (2007). Scheduling hybrid flow shop for minimizing total weight completion time. *Proceedings of the IEEE International Conference on Automation and Logistics*, ICAL 2007 : 809-813.

[82] Garey M.R., and D.S. Johnson. (1978) "Strong" NP-completeness results : motivation, examples, and implications. *Journal of the Association for Computing Machinery*, 25(3) :499-508.

[83] Garey, M.R., and D.S. Johnson. (1979). Computers and Intractibility : A Guide to the Theory of NP-Completeness. *W.H. Freeman and co.*

[84] Gharbi, A., and M. Haouari. (2005). Optimal parallel machines scheduling with availability constraints. *Discrete Applied Mathematics,* 148(1) : 63-87.

[85] Glassey, C. Roger, and W. Willie Weng. (1991). Dynamic batching heuristic for simultaneous processing. *IEEE Transactions on Semiconductor Manufacturing*, 4(2) : 77-82.

[86] Graham, R.L., E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. (1979). Optimization and approximation in deterministic sequencing and scheduling : a survey. *Annals of Discrete Mathematics*, 5 : 287-326.

[87] Guinet, A., M. M. Solomon, P. K. Kedia, and A. Dussauchoy. (1996). A computational study of heuristics for two-stage flexible flowshops. *International Journal of Production Research*, 34(5) : 1399-1415.

[88] Gupta, A. K., V. K. Ganesan, and A. I. Sivakumar. (2004). Hot lot management : Minimizing cycle time in batch processes. *IEEE International Engineering Management Conference* 3 : 1217-1221.

[89] Gupta, A. K., and A. I. Sivakumar. (2006). Optimization of due-date objectives in scheduling semiconductor batch manufacturing. *International Journal of Machine Tools and Manufacture*, 46(12-13) : 1671-1679.

[90] Gupta, J. N. D. (1988). Two-stage, hybrid flowshop scheduling problem. *Journal of the Operational Research Society*, 39(4) : 359-364.

[91] Gupta, J. N. D., A. M. A. Hariri, and C. N. Potts. (1997). Scheduling a two-stage hybrid flow shop with parallel machines at the first stage. *Annals of Operations Research*, 69 : 171-191.

[92] Gupta, J. N. D., and E. F. Stafforned Jr. (2006). Flowshop scheduling research after five decades. *European Journal of Operational Research*, 169(3) : 699-711.

[93] Gupta, J. N. D., and E. A. Tunc. (1998). Minimizing tardy jobs in a two-stage hybrid flowshop. *International Journal of Production Research*, 36(9) : 2397-2417.

[94] Gupta, U. I., D. T. Lee, and J. Y.-T. Leung. (1982) Efficient algorithms for interval graphs and circular-arc graphs. *Networks*, 12(4) :459-467.

[95] Guschinskaya, O., A. Dolgui, N. Guschinsky, and G. Levin. (2009). Minimizing makespan for multi-spindle head machines with a mobile table. *Computers & Operations Research*, 36(2) : 344-357.

[96] Hall, L.A., A.S. Schulz, D.B. Shmoys and J. Wein. (1997). Scheduling to minimize average completion time : offline and online algorithms. *Mathematics of Operations Research*, 22 : 513–544.

[97] Hall, L.A., and D.B. Shmoys. (1989). Approximation schemes for constrained scheduling problems. *Proceedings of the 30th annual IEEE symposium on foundations of computer science*, 134–139.

[98] Halldórsson, M. M., and H. Shachnai. (2008). Batch coloring flat graphs and thin. *Lecture Notes in Computer Science*, 5124 LNCS : 198-209.

[99] Haouari, M., and T. Daouas. (1999). Optimal scheduling of the 3-machine assembly-type flow shop. *RAIRO Recherche Operationnelle*, 33(4) : 439-445.

[100] Haouari, M., and L. Hidri. (2008). On the hybrid flowshop scheduling problem. *International Journal of Production Economics*, 113(1) : 495-497.

[101] Haouari, M., L. Hidri, and A. Gharbi. (2006). Optimal scheduling of a two-stage hybrid flow shop. *Mathematical Methods of Operations Research*, 64(1) : 107-124.

[102] Hariri, A. M. A., and C. N. Potts. (1997). A branch and bound algorithm for the two-stage assembly scheduling problem. *European Journal of Operational Research*, 103(3) : 547-556.

[103] He, C., Y. Lin, and J. Yuan. (2007). Bicriteria scheduling on a batching machine to minimize maximum lateness and makespan. *Theoretical Computer Science*, 381(1-3) : 234-240.

[104] He, L. -M, and S. -J Sun. (2007). A hybrid two-stage flexible flowshop scheduling problem with m identical parallel machines and a burn-in processor separately. *Journal of Shanghai University*, 11(1) : 33-38.

[105] He, L., S. Sun, and R. Luo. (2007). A hybrid two-stage flowshop scheduling problem. *Asia-Pacific Journal of Operational Research*, 24(1) : 45-56.

[106] Hmida, A. B., M. -J Huguet, P. Lopez, and M. Haouari. (2007). Adaptation of discrepancy-based methods for solving hybrid flow shop problems. *Proceedings - ICSSSM'06 : 2006 International Conference on Service Systems and Service Management 2* : 1120-1125

[107] Hochbaum, D. S., and D. Landy. (1997). Scheduling semiconductor burn-in operations to minimize total flowtime. *Operations research*, 45(6) : 874-885.

[108] Hoogeveen, H., and S. Van De Velde. (1998). Scheduling by positional completion times : Analysis of a two-stage flow shop problem with a batching machine. *Mathematical Programming*, 82 : 273-289.

[109] Ikura, Y., and M. Gimple. (1986). Efficient scheduling algorithms for a single batch processing machine. *Operations Research Letters*, 5(2) : 61-65.

[110] Jin, Z., Z. Yang, and T. Ito. (2006). Metaheuristic algorithms for the multistage hybrid flowshop scheduling problem. *International Journal of Production Economics*, 100(2) : 322-334.

[111] Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1) : 61-68.

[112] Jolai, F. (2005). Minimizing number of tardy jobs on a batch processing machine with incompatible job families. *European Journal of Operational Research*, 162(1) : 184-190.

[113] Jolai Ghazvini, F., and L. Dupont. (1998). Minimizing mean flow times criteria on a single batch processing machine with non-identical jobs sizes. *International Journal of Production Economics*, 55(3) : 273-280.

[114] Jouglet, A., C. Oğuz, and M. Sevaux. (2009). Hybrid flow-shop : A memetic algorithm using constraint-based scheduling for efficient search. *Journal of Mathematical Modelling and Algorithms* : 1-22

[115] Kashan, A. H., and B. Karimi. (2008). Scheduling a single batch-processing machine with arbitrary job sizes and incompatible job families : An ant colony framework. *Journal of the Operational Research Society*, 59(9) : 1269-1280.

[116] Kashan, A. H., and B. Karimi. (2009). An improved mixed integer linear formulation and lower bounds for minimizing makespan on a flow shop with batch processing machines. *International Journal of Advanced Manufacturing Technology*, 40(5-6) : 582-594.

[117] Kashan, A. H., B. Karimi, and S. M. T. Fatemi Ghomi. (2009). A note on minimizing makespan on a single batch processing machine with nonidentical job size. *Theoretical Computer Science*, 410(27-29) : 2754-2758.

[118] Kashan, A. H., B. Karimi, and M. Jenabi. (2008). A hybrid genetic heuristic for scheduling parallel batch processing machines with arbitrary job sizes. *Computers & Operations Research*, 35(4) : 1084-1098.

[119] Kashan, A. H., B. Karimi, and F. Jolai. (2006). Minimizing makespan on a single batch processing machine with non-identical job sizes : A hybrid genetic approach. *Lecture Notes in Computer Science*, 3906(LNCS) : 135-146.

[120] Kempf, K. G., R. Uzsoy, and C.S. Wang. (1998). Scheduling a single batch processing machine with secondary resource constraints. *Journal of Manufacturing Systems*, 17(1) : 37-51.

[121] Kierstead, H. A., and J. Qin. (1995). Coloring interval graphs with first-fit. *Discrete Mathematics*, 144(1-3) : 47-57.

[122] Kim, Y. -D, B. -J Joo, and J. -H Shin. (2009). Heuristics for a two-stage hybrid flow-shop scheduling problem with ready times and a product-mix ratio constraint. *Journal of Heuristics*, 15(1) : 19-42.

[123] Kis, T., and E. Pesch. (2005). A review of exact solution methods for the non-preemptive multiprocessor flowshop problem. *European Journal of Operational Research*, 164(3) : 592-608.

[124] Koh, S. -G, P. -H Koo, J. -W Ha, and W. -S Lee. (2004). Scheduling parallel batch processing machines with arbitrary job sizes and incompatible job families. *International Journal of Production Research*, 42(19) : 4091-4107.

[125] Koh, S. -G, P. -H Koo, D. -C Kim, and W. -S Hur. (2005). Scheduling a single batch processing machine with arbitrary job sizes and incompatible job families. *International Journal of Production Economics*, 98(1) : 81-96.

[126] Koulamas, C., and G. Kyparisis. (2001). The three-stage assembly flowshop scheduling problem. *Computers & Operations Research*, 28(7) : 689-704.

[127] Kubiak, W., and F. Jolai Ghazvini. (1997). Minimizing earliness/tardiness criteria on a batch processing machine with job families. In *Second Annual International Conference on Industrial Engineering*, 2 :785-790.

[128] Kumar Manjeshwar, P., P. Damodaran, and K. Srihari. (2009). Minimizing makespan in a flow shop with two batch-processing machines using simulated annealing. *Robotics and Computer-Integrated Manufacturing*, 25(3) : 667-679.

[129] Kurz, M. E., and R. G. Askin. (2004). Scheduling flexible flow lines with sequence-dependent setup times. *European Journal of Operational Research*, 159(1) : 66-82

[130] Kurz, M. E., and S. J. Mason. (2008). Minimizing total weighted tardiness on a batch-processing machine with incompatible job families and job ready times. *International Journal of Production Research*, 46(1) : 131-151.

[131] Lawler, E.L., J.K. Lenstra,A.H.G. Rinnooy Kan, and D.B. Shmoys. (1993) Sequencing and scheduling : algorithms and complexity. In *Logistics of Production and Inventory.* Amsterdam, Netherlands. : 445-522.

[132] Lawler, E.L., and J.M. Moore. (1969) A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16 : 77–84.

[133] Lee, C.Y., T. C. E. Cheng, and B. M. T. Lin. (1993). Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem. *Management Science*, 39(5) : 616-625.

[134] Lee, G. -C, and Y. -D Kim. (2004). A branch-and-bound algorithm for a two-stage hybrid flowshop scheduling problem minimizing total tardiness. *International Journal of Production Research*, 42(22) : 4731-4743.

[135] Lee, G. -C, Y. -D Kim, and S. -W Choi. (2004). Bottleneck-focused scheduling for a hybrid flowshop. *International Journal of Production Research*, 42(1) : 165-181.

[136] Lee, C.Y., R. Uzsoy and L.A. Martin Vega. (1992). Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40 : 764–775

[137] Lee, C. -Y, and R. Uzsoy. (1999). Minimizing makespan on a single batch processing machine with dynamic job arrivals. *International Journal of Production Research*, 37(1) : 219-236.

[138] Li, C.L., and C.Y. Lee. (1997). Scheduling with agreeable release times and due dates on a batch processing machine. *European Journal of Operational Research*, 96(3) : 564-569.

[139] Li, S., G. Li, X. Wang, and Q. Liu. (2005). Minimizing makespan on a single batching machine with release times and non-identical job sizes. *Operations Research Letters*, 33(2) : 157-164.

[140] Li, S., G. Li, and S. Zhang. (2005). Minimizing makespan with release times on identical parallel batching machines. *Discrete Applied Mathematics*, 148(1) : 127-134.

[141] Li, L., and F. Qiao. (2008). ACO-based scheduling for a single batch processing machine in semiconductor manufacturing. *4th IEEE Conference on Automation Science and Engineering*, CASE 2008 : 85-90.

[142] Li, L., F. Qiao, and Q. Wu. (2008) ACO-based scheduling of parallel batch processing machines with incompatible job families to minimize total weighted tardiness. *Lecture Notes in Computer Science*, 5217 LNCS : 219-226

[143] Lili, L., and Z. Feng. (2008). Minimizing number of tardy jobs on a batch processing machine with incompatible job families. *Proceedings - ISECS International Colloquium on Computing, Communication, Control, and Management, CCCM 2008*, 3 : 277-280.

[144] Li, S., and J. Yuan. (2008). Parallel-machine parallel-batching scheduling with family jobs and release dates to minimize makespan. *Journal of Combinatorial Optimization*, In Press.

[145] Liao, C. -J, and L. -M Liao. (2008). Improved MILP models for two-machine flowshop with batch processing machines. *Mathematical and Computer Modelling*, 48(7-8) : 1254-1264.

[146] Linn, R., and W. Zhang. (1999). Hybrid flow shop scheduling : A survey. *Computers and Industrial Engineering*, 37(1) : 57-61.

[147] Liu, Z., and T. C. E. Cheng. (2005). Approximation schemes for minimizing total (weighted) completion time with release dates on a batch machine. *Theoretical Computer Science*, 347 : 288-298,

[148] Liu, L. L., C. T. Ng, and T. C. E. Cheng. (2007). Scheduling jobs with agreeable processing times and due dates on a single batch processing machine. *Theoretical Computer Science*, 374(1-3) : 159-169.

[149] Liu, L. L., C. T. Ng, and T. C. E. Cheng. (2009). Scheduling jobs with release dates on parallel batch processing machines. *Discrete Applied Mathematics*, 157(8) : 1825-1830.

[150] Liu, Z., and W. Yu.(2000) Scheduling one batch processor subject to job release dates. *Discrete Applied Mathematics*, 105(1-3) : 129-136.

[151] Liu, Z., J. Yuan, and T. C. E. Cheng. (2003). On scheduling an unbounded batch machine. *Operations Research Letters*, 31(1) : 42-48.

[152] Lopez, P., and F. Roubellat. (2001). Ordonnancement de la production. *Hermes*, Paris.

[153] Lu, L., T. C. E. Cheng, J. Yuan, and L. Zhang. (2009). Bounded single-machine parallel-batch scheduling with release dates and rejection. *Computers & Operations Research*, 36(10) : 2748-2751.

[154] Lu, L., and J. Yuan. (2008). Unbounded parallel batch scheduling with job delivery to minimize makespan. *Operations Research Letters*, 36(4) : 477-480.

[155] Lu, L., L. Zhang, and J. Yuan. (2008). The unbounded parallel batch machine scheduling with release dates and rejection to minimize makespan. *Theoretical Computer Science*, 396(1-3) : 283-289.

[156] Malve, S., and R. Uzsoy. (2007). A genetic algorithm for minimizing maximum lateness on parallel identical batch processing machines with dynamic job arrivals and incompatible job families. *Computers & Operations Research*, 34(10) : 3016-3028.

[157] Mathirajan, M., and A. I. Sivakumar. (2006). A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. *International Journal of Advanced Manufacturing Technology*, 29(9-10) : 990-1001.

[158] Mathirajan, M., and A. I. Sivakumar. (2006). Minimizing total weighted tardiness on heterogeneous batch processing machines with incompatible job families. *International Journal of Advanced Manufacturing Technology*, 28(9) : 1038-1047.

[159] Mathirajan, M., A. I. Sivakumar, and V. Chandru. (2004). Scheduling algorithms for heterogeneous batch processors with incompatible job-families. *Journal of Intelligent Manufacturing*, 15(6) : 787-803.

[160] Melouk, S., P. Damodaran, and P.Y. Chang. (2004). Minimizing makespan for single machine batch processing with non-identical job sizes using simulated annealing. *International Journal of Production Economics*, 87(2) : 141-147.

[161] Mehta, S.V., and R. Uzsoy. (1998). Minimizing total tardiness on a batch processing machine with incompatible job families. *I.I.E. Transaction on Scheduling and Logistics*, 31 : 165-178.

[162] Mirsanei, H. S., B. Karimi, and F. Jolai. (2009). Flow shop scheduling with two batch processing machines and nonidentical job sizes. *International Journal of Advanced Manufacturing Technology* : 1-20.

[163] Mönch, L., H. Balasubramanian, J.W. Fowler, and M.E. Pfund. (2005). Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. *Computers & Operations Research*, 32 : 2731-2750

[164] Mönch, L., R. Unbehaun, and Y. I. Choung. (2006). Minimizing earliness-tardiness on a single burn-in oven with a common due date and maximum allowable tardiness constraint. *OR Spectrum*, 28(2) : 177-198.

[165] Moursli, O., and Y. Pochet. (2000). Branch-and-bound algorithm for the hybrid flowshop. *International Journal of Production Economics*, 64(1) : 113-125.

[166] Negenman, E. G. (2001). Local search algorithms for the multiprocessor flow shop scheduling problem. *European Journal of Operational Research*, 128(1) : 147-158.

[167] Néron, E., P. Baptiste, and J. N. D. Gupta. (2001). Solving hybrid flow shop problem using energetic reasoning and global operations. *Omega,* 29(6) : 501-511

[168] Neumann, K., C. Schwindt, and N. Trautmann. (2002). Advanced production scheduling for batch plants in process industries. *OR Spectrum*, 24(3) : 251-279.

[169] Ng, C. T., M. Y. Kovalyov, and T. C. E. Cheng. (2010). A simple FPTAS for a single-item capacitated economic lot-sizing problem with a monotone cost structure. *European Journal of Operational Research*, 200(2) : 621-624.

[170] Nong, Q. Q., T. C. E. Cheng, and C. T. Ng. (2008). An improved on-line algorithm for scheduling on two unrestrictive parallel batch processing machines. *Operations Research Letters*, 36(5) : 584-588.

[171] Nong, Q. Q., C. T. Ng, and T. C. E. Cheng. (2008). The bounded single-machine parallel-batching scheduling problem with family jobs and release dates to minimize makespan. *Operations Research Letters*, 36(1) : 61-66.

[172] Nong, Q., J. Yuan, R. Fu, L. Lin, and J. Tian. (2008). The single-machine parallel-batching on-line scheduling problem with family jobs to minimize makespan. *International Journal of Production Economics*, 111(2) : 435-440.

[173] Nowicki, E., and C. Smutnicki. (1998). The flow shop with parallel machines : A tabu search approach. *European Journal of Operational Research*, 106(2-3) : 226-253.

[174] Oğuz, C., and M.F. Ercan. (2005). A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks. *Journal of Scheduling,* 8(4) : 323–351.

[175] Oulamara, A., and G. Finke. (2001). Flowshop problems with two batch processing machines. *International Journal of Mathematical Algorithms*, 2 : 269-287.

[176] Oulamara, A., G. Finke, and A. Kamgaing Kuiteing. 2009. Flowshop scheduling problem with a batching machine and task compatibilities. *Computers & Operations Research*, 36(2) : 391-401

[177] Oulamara, A., M. Kovalyov, and G. Finke. (2005). Scheduling a no-wait flow shop containing unbounded batching machines. *IIE Transactions*, 37(8) : 685-696.

[178] Pemmaraju, S. V., R. Raman, and K. Varadarajan. (2004). Buffer minimization using max-coloring. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, 15 : 555-564.

[179] Perez, I.C., J.W. Fowler, and W.M. Carlyle. (2005). Minimizing total weighted tardiness on a single batch process with incompatible job families. *Computers & Operation Research*, 32 : 327-341.

[180] Pinedo M. (1995). Scheduling theory, algorithms, and systems. *Prentice-Hall, Englewood Cliffs, NJ.*

[181] Poon, C. K., and W.C. Yu. (2005). A flexible on-line scheduling algorithm for batch machine with infinite capacity. *Annals of Operations Research*, 133(1-4) : 175-181.

[182] Poon C.K., and W.C. Yu. (2005). On-line scheduling algorithms for a batch machine with finite capacity. *Journal of Combinatorial Optimization*, 9 : 167–186

[183] Poon C.K., and P. Zhang. (2004). Minimizing makespan in batch machine scheduling. *Algorithmica*, 39 : 155–174.

[184] Portmann, M.-C., A. Vignier, D. Dardilhac, and D. Dezalay. (1998). Branch and bound crossed with GA to solve hybrid flow shops. *European Journal of Operational Research* 107(2) : 389–400.

[185] Potts, C. N., and M. Y. Kovalyov. (2000). Scheduling with batching : A review. *European Journal of Operational Research* 120(2) : 228-249.

[186] Potts, C.N.,S.V. Sevast'janov, V.A. Strusevich, L.N. Van Wassenhove, and C.M. Zwaneveld. (1995). The two-stage assembly scheduling problem : complexity and approximation. *Operations Research*, 43, pp. 346-355.

[187] Potts, C. N., V. A. Strusevich, and T. Tautenhahn. (2001). Scheduling batches with simultaneous job processing for two-machine shop problems. *Journal of Scheduling*, 4(1) : 25-51.

[188] Potts, C.N., and L.N. Van Wassenhove. (1992). Integrating scheduling with batching and lotsizing : A review of algorithms and complexity. *Journal of the Operations Research Society*, 42 : 395-406.

[189] Qi, X., S. Zhou, and J. Yuan. (2009). Single machine parallel-batch scheduling with deteriorating jobs. *Theoretical Computer Science*, 410(8-10) : 830-836.

[190] Ridouard, F., P. Richard, and P. Martineau. (2008). On-line scheduling on a batch processing machine with unbounded batch size to minimize the makespan. *European Journal of Operational Research*, 189(3) : 1327-1342.

[191] Rinnooy Kan, A.H.G. (1976). Machine scheduling problems : classification, complexity and computation. *Nijhoff, The Hague.*

[192] Shubin, X., and J. C. Bean. (2007). A genetic algorithm for scheduling parallel non-identical batch processing machines. *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling*, CI-Sched 2007 : 143-150.

[193] Shuguang L., L. Guojun, W. Xiaoli and L. Qiming. (2005). Minimizing makespan on a single batching machine with release times and non-identical job sizes. *Operations Research Letters*, 33 : 157–164.

[194] Smith, W.E. (1956). Various optimizers for single stage production. *Naval Research Logistics Quarterly,* 3 : 59–66.

[195] Soewandi, H., and S. E. Elmaghraby. (2001). Sequencing three-stage flexible flowshops with identical machines to minimize makespan. *IIE Transactions*, 33(11) : 985-993.

[196] Srinivasa Raghavan, N. R., and M. Venkataramana. (2007). Scheduling parallel batch processors with incompatible job families using ant colony optimization. *2006 IEEE International Conference on Automation Science and Engineering*, CASE : 507-512.

[197] Su, L.H. (2003). A hybrid two-stage flowshop with limited waiting time constraints. *Computers and Industrial Engineering*, 44(3) : 409-424.

[198] Sun, X., K. Morizawa, and H. Nagasawa. (2003). Powerful heuristics to minimize makespan in fixed, 3-machine, assembly-type flowshop scheduling. *European Journal of Operational Research*, 146(3) : 498-516.

[199] Sung, C.S., and Y. I. Choung. (2000). Minimizing makespan on a single burn-in oven in semiconductor manufacturing. *European Journal of Operational Research*, 120(3) : 559-574.

[200] Sung, C. S., Y. I. Choung, J. M. Hong, and Y. H. Kim. (2002). Minimizing makespan on a single burn-in oven with job families and dynamic job arrivals. *Computers & Operations Research*, 29(8) : 995-1007.

[201] Sung, C.S., and Y.H. Kim. (2002). Minimizing makespan in a two-machine flowshop with dynamic arrivals allowed. *Computers & Operations Research*, 29 : 275–294.

[202] Sung, C.S., Y.H. Kim, and S.H. Yoon. (2000) A problem reduction and decomposition approach for scheduling a flowshop of batch processing machines. *European Journal of Operational Research*, 121(1) : 179-192.

[203] Tang, L., and H. Gong. (2009). The coordination of transportation and batching scheduling. *Applied Mathematical Modelling*, 33(10) : 3854-3862.

[204] Tang, L., H. Xuan, and J. Liu. (2006). A new lagrangian relaxation algorithm for hybrid flowshop scheduling to minimize total weighted completion time. *Computers & Operations Research*, 33(11) : 3344-3359.

[205] Tang, L., and Y. Zhao. (2008). Scheduling a single semi-continuous batching machine. *Omega*, 36(6) : 992-1004.

[206] Tangudu, S. K., and M. E. Kurz. (2006). A branch and bound algorithm to minimise total weighted tardiness on a single batch processing machine with ready times and incompatible job families. *Production Planning and Control*, 17(7) : 728-741.

[207] Tian J., R. Fu, and J. Yuan. (2007). On-line scheduling with delivery time on a single batch machine. *Theoretical Computer Science*, 374 : 49–57.

[208] Tozkapan, A., O. Kirca, and C. -S Chung. (2003). A branch and bound algorithm to minimize the total weighted flowtime for the two-stage assembly scheduling problem. *Computers & Operations Research*, 30(2) : 309-320.

[209] Uzsoy, R. (1994). Scheduling a single batch processing machine with non-identical job sizes. *International Journal of Production Research*, 32 : 1615-1638.

[210] Uzsoy, R. (1995). Scheduling batch processing machines with incompatible job families. *International Journal of Production Research*, 33 :2635-2708.

[211] Uzsoy, R., and Y. Yang. (1997). Minimizing total weighted completion time on a single batch processing machine. *Production and Operations Management*, 6 : 57-73.

[212] Van Der Zee, D. J. (2004). Dynamic scheduling of batch servers with compatible product families. *International Journal of Production Research*, 42(22) : 4803-4826.

[213] Van Der Zee, D. J. (2007). Dynamic scheduling of batch-processing machines with non-identical product sizes. *International Journal of Production Research*, 45(10) : 2327-2349.

[214] Vandevelde, A., H. Hoogeveen, C. Hurkens, and J. K. Lenstra. (2005). Lower bounds for the head-body-tail problem on parallel machines : A computational study of the multiprocessor flow shop. *INFORMS Journal on Computing*, 17(3) : 305-320.

[215] Vignier A. (1997). Contribution à la résolution des problèmes d'ordonnancement de type monogamme, multimachine ("Flow-shop hybride"). *Thèse de Doctorat, Université de Tours, France.*

[216] Vignier, A., J.C. Billaut, and C. Proust. (1999). Les problèmes d'ordonnancement de type flow-shop hybride : État de l'art. *RAIRO - Operations Research*, 33(2) : 117-183

[217] Wang, H. -M, P. -C Chang, and F. -D Chou. (2007). A hybrid forward/backward approach for single batch scheduling problems with non-identical job sizes. *Journal of the Chinese Institute of Industrial Engineers*, 24(3) : 191-199.

[218] Wang, C. -S, and R. Uzsoy. (2002). A genetic algorithm to minimize maximum lateness on a batch processing machine. *Computers & Operations Research*, 29(12) : 1621-1640.

[219] Webster, S., and K.R. Baker. (1995). Scheduling groups of jobs on a single machine. *Operations Research*, 43 : 692-704.

[220] Yang, Y., S. Kreipl, and M. Pinedo. (2000). Heuristics for minimizing total weighted tardiness in flexible flow shops. *Journal of Scheduling*, 3(2) : 89-108.

[221] Yong, Z., and Q. Changhua. (2008). Genetic algorithm application to the hybrid flow shop scheduling problem. *Proceedings of 2008 IEEE International Conference on Mechatronics and Automation*, ICMA 2008 : 649-653,

[222] Yuan, J., S. Li, J. Tian, and R. Fu. (2009). A best on-line algorithm for the single machine parallel-batch scheduling with restricted delivery times. *Journal of Combinatorial Optimization*, 17(2) : 206-213.

[223] Yuan, J. J., Z.H. Liu , C.T. Ng, and T.C.E. Cheng. (2004). The unbounded single machine parallel batch scheduling problem with family jobs and release dates to minimize makespan. *Theoretical Computer Science*, 320 : 199–212.

[224] Yuan, J., X. Qi, L. Lu, and W. Li. (2008). Single machine unbounded parallel-batch scheduling with forbidden intervals. *European Journal of Operational Research*, 186(3) : 1212-1217.

[225] Zhang, Y., C. Bai, Q. Bai, and J. Xu. (2007). Duplicating in batch scheduling. *Journal of Industrial and Management Optimization*, 3(4) : 685-692.

[226] Zhang, G., X. Cai, C. -Y Lee, and C. K. Wong. (2001). Minimizing makespan on a single batch processing machine with nonidentical job sizes. *Naval Research Logistics*, 48(3) : 226-240.

[227] Zhang G., X. Cai, and C. Wong. (2001). On-line algorithms for minimizing makespan on batch processing machines. *Naval Research Logistics*, 48 : 241–258.

[228] Zhang, Y., and Z. Cao. (2007). An asymptotic PTAS for batch scheduling with nonidentical job sizes to minimize makespan. *Lecture Notes in Computer Science*, 4616 LNCS : 44-51.

[229] Zhang, Y., and Z. Cao. (2008). An asymptotic PTAS for batch scheduling with nonidentical job sizes to minimize makespan. *Journal of Combinatorial Optimization*, 16(2) : 119-126.

[230] Zhang, Y., Z. Cao, and Q. Bai. (2005). A PTAS for scheduling on agreeable unrelated parallel batch processing machines with dynamic job arrivals. *Lecture Notes in Computer Science*, 3521 : 162-171.

[231] Zhang, W. -G, H. -P Chen, D. Lu, and H. Shao. (2008). A novel differential evolution algorithm for a single batch-processing machine with non-identical job sizes. *Proceedings - 4th International Conference on Natural Computation, ICNC 2008*, 6 : 447-451.

[232] Zhao, H., and G. Li. (2008). Unbounded batch scheduling with a common due window on a single machine. *Journal of Systems Science and Complexity*, 21(2) : 296-303.

INSTITUT NATIONAL
POLYTECHNIQUE
DE LORRAINE

AUTORISATION DE SOUTENANCE DE THESE
DU DOCTORAT DE L'INSTITUT NATIONAL
POLYTECHNIQUE DE LORRAINE

o0o

VU LES RAPPORTS ETABLIS PAR :
Monsieur Pierre LOPEZ, Chargé de Recherche, Groupe MOGISA, LAAS-CNRS, Toulouse

Monsieur Bernard PENZ, Professeur, Laboratoire G-SCOP, INPG, Grenoble

Le Président de l'Institut National Polytechnique de Lorraine, autorise :

Monsieur BELLANGER Adrien

NANCY BRABOIS
2, AVENUE DE LA
FORET-DE-HAYE
BOITE POSTALE 3
F - 54501
VANDŒUVRE CEDEX

à soutenir devant un jury de l'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE, une thèse intitulée :

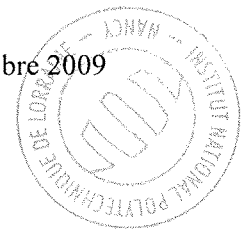
« Ordonnancement sur les machines à traitement par batches et contraintes de compatibilité »

en vue de l'obtention du titre de :

DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

en « Informatique »

Fait à Vandoeuvre, le 12 novembre 2009
Le Président de l'I.N.P.L.,
F. LAURENT



TEL. 33/03.83.59.59.59
FAX. 33/03.83.59.59.55

Ordonnancement sur les machines à traitement par batches et contraintes de compatibilité.

Résumé

Dans cette thèse, nous avons traité les problèmes d'ordonnancement d'ateliers de type flowshop hybride à deux étages avec machines à traitement par batches sur le second étage et compatibilité entre les tâches.Les durées opératoires des tâches sont données par des intervalles, et les tâches sont dites compatibles si elles partagent une même durée d'exécution. Pour le problème de minimisation de la date de fin d'ordonnancement de ce type d'atelier, nous avons développé 6 heuristiques à performances garanties. D'après les expériences réalisées, ces heuristiques sont efficaces sur de grandes instances. Pour les petites instances, nous avons présenté deux méthodes exactes de type procédures par séparation évaluation qui permettent de résoudre des instances de 20 tâches. Nous avons également développé un schéma d'approximation polynomial (PTAS) utilisable lorsque les durées d'exécution sur le premier étage sont identiques. En complément de ces travaux, nous avons également étudié d'autres problèmes de minimisation de critères réguliers sur une machine à traitement par batches. Nous avons développé des algorithmes de programmation dynamiques pseudo-polynomiaux pour les problèmes de minimisation de la somme des dates de fin d'exécution et pour les problèmes avec dates de fin souhaitées. Afin de compléter ces résultats de complexité, nous avons montré la NP-complétude des problèmes avec dates de fin souhaitées.

Mots-clés: Ordonnancement, batch, compatibilité, flowshop hybride.

Scheduling batching machines with compatibility constraints.

Abstract

This thesis deals with 2-stages hybrid flowshop scheduling problems with batching machines on the second stage and compatibility constraints. The processing times of tasks are given by intervals and tasks are compatible if they share a same second stage processing time. We developped 6 heuristics with worth-case analysis for the makespan minimization problem. The experimental results show that these heuristics give good schedules with an average gap of 1% on 200 task instances. For small instances, we presented 2 exact methods, Branch & Bounds, which solves up to 20 task instances. For the particular case with identical processing times on first stage and uniform processing time intervals we developped a Polynomial Time Approximation Scheme (PTAS). The second part of this thesis deal with scheduling problems on one batching machine with infinite capacity and regular criteria minimization. We developped pseudo-polynomial dynamic programming algorithm for minimization total completion time, maximal lateness and total tardiness. Finally we show the NP-completeness of problems with due dates.

Keywords: Scheduling, batching machines, graph compatibility, hybrid flowshop.