RAIRO-Oper. Res. 43 (2009) 421–436
DOI: 10.1051/ro/2009024

RAIRO Operations Research
www.rairo-ro.org

MÉTHODE HEURISTIQUE POUR LE PROBLÈME DE FLOW SHOP HYBRIDE AVEC MACHINES DÉDIÉES

Najoua Dridi[1], Hatem Hadda[2] et Sonia Hajri-Gabouj[2]

Résumé. Dans ce papier, nous traitons le problème de minimisation du makespan dans un flow shop hybride à deux étages avec machines dédiées. En premier lieu, nous présentons des propriétés de base, un ensemble de bornes inférieures et deux cas polynomiaux. En second lieu, nous proposons une nouvelle heuristique qui exploite ces propriétés, et cherche à placer les jobs, en tenant compte pour chaque instance du problème, de la valeur de la borne inférieure. La dernière partie de ce travail présente les résultats expérimentaux d'une étude comparative avec une heuristique de la littérature. L'analyse de ces résultats permet d'apprécier la qualité de notre proposition.

Mots Clés. Ordonnancement, flow shop hybride, machines dédiées, bornes inférieures, cas polynomiaux, heuristiques.

Abstract. In this paper we deal with the two-stage hybrid flow shop with dedicated machines. The objective is to minimize the makespan. We first provide some basic properties, a set of lower bounds and two polynomial cases. We then propose a new heuristic based on the established properties. The heuristic tries to sequence jobs in such a way that the obtained makespan corresponds to the lower bound. In the last part, we present the results of a computational experience comparing our heuristic with another heuristic found in the literature. The analysis of those results allows us to appreciate the quality of our proposition.

Keywords. Scheduling, hybrid flow shop, dedicated machines, lower bounds, polynomial cases, heuristics.

Classification Mathématique. 90B35, 90B30.

Reçu le 26 janvier, 2008. Accepté le 17 juin, 2009.

[1] Unité de recherche OASIS, ENIT, B.P. 37, Le belvédère 100, Tunis, Tunisie ; najoua.dridi@enit.rnu.tn

[2] Unité de Recherche URAII, INSAT-TUNIS, Centre Urbain Nord B.P. No. 676, 1080 Tunis Cedex, Tunisie ; hatem.hadda@esti.rnu.tn ; sonia.gabouj@insat.rnu.tn

Article published by EDP Sciences

1. Introduction

Ce papier traite un cas particulier du problème de flow shop hybride à deux étages, dans lequel chaque job nécessite exactement deux opérations. Le premier étage est constitué d'une seule machine, quant au second, il comprend m machines dédiées. Tous les jobs passent d'abord, par la seule machine du premier étage (M_c : machine commune). Ensuite, et suivant le type du job, la deuxième opération est prise en charge par l'une des m machines du second étage. L'objectif est la minimisation de la date d'achèvement du dernier job (le makespan). Ce problème est noté $F_{m+1}|T = m|C_{\max}$ [9] où T désigne le nombre de types de jobs à réaliser.

Le problème étudié constitue la configuration la plus simple du flow shop hybride avec machines dédiées. Un exemple d'application peut être trouvé en industrie pharmaceutique, où les produits passent d'abord par un ensemble de machines communes (Doseur, mélangeur, ...). Suite à ces étapes, chaque produit suit une ligne de conditionnement spécifique, suivant sa forme finale (pilule, poudre, gélule, ...). D'autres exemples peuvent être rencontrés dans la production de meubles [12], la fabrication d'étiquettes autocollantes [7] ...

Dans le cas d'un seul type de job ($m = 1$), le problème est alors un flow shop à deux machines, et l'algorithme de Johnson le résout en un temps polynomial [4]. Hormis ce cas, la configuration la plus simple de $F_{m+1}|T = m|C_{\max}$ (pour $m = 2$) est NP-difficile au sens fort [6].

Plusieurs travaux ont traité le cas particulier avec deux types de jobs. Dans [11] la programmation dynamique est utilisée dans une approche de résolution du problème $F_3|T = 2|C_{\max}$. Les auteurs ont également identifié quatre cas polynomiaux. Plus récemment dans [8], plusieurs bornes inférieures sont proposées pour le problème $F_3|T = 2|C_{\max}$.

Dans [9], le problème $F_3|T = 2|C_{\max}$, avec deux machines dédiées au premier étage et une seule machine au second, est étudié. Les auteurs ont développé quelques bornes inférieures et deux cas polynomiaux. Ils ont également proposé une heuristique de résolution et étudié son comportement au pire des cas. Dans cette approche, ils traitent chaque type de jobs à part, considérant ainsi, deux problèmes à deux machines qu'ils résolvent par l'algorithme de Johnson. La solution du problème initial est obtenue en suivant l'ordre croissant des dates de fin des jobs au premier étage. À la fin, tous les jobs d'un même type suivent l'ordre de Johnson.

Un rapport étroit existe entre le problème traité dans cet article et celui rencontré dans les ateliers d'assemblage, où m machines composent le premier étage et une seule machine est au second et dans lesquels, un job nécessite une opération par machine. Les m premières opérations peuvent s'effectuer en parallèle, la dernière opération ne peut commencer que lorsque les m premières sont terminées. Le problème est noté $A_m||C_{\max}$ lorsque l'objectif est la minimisation du makespan [10]. Le cas le plus simple ($m = 2$) de $A_m||C_{\max}$, noté 3MAF, est NP-difficile au sens fort [5].

Le problème miroir de 3MAF est défini plus tard [3]. C'est le flow shop de désassemblage à trois machines (3MDF). Une seule machine compose le premier

étage, et deux machines sont au second. Un job nécessite trois opérations, une fois l'opération sur la machine du premier étage terminée, les deux opérations sur les machines du second étage peuvent commencer. Les auteurs ont montré que ce problème est équivalent au problème 3MAF.

Nous généralisons le problème de désassemblage 3MDF, pour obtenir le problème que nous notons $D_m||C_{\max}$, comportant une seule machine au premier étage et m machines au second. Il correspond au problème miroir de $A_m||C_{\max}$. Le lemme suivant généralise l'équivalence prouvée dans [3] entre les problèmes 3MAF et 3MDF.

Lemme 1.1. *En inversant l'ordre d'une solution du problème $A_m||C_{\max}$, nous obtenons une solution du problème $D_m||C_{\max}$ avec la même valeur du makespan.*

Démonstration. Identique à la preuve donnée pour le cas $m = 2$, dans [3]. □

Le problème $F_{m+1}|T = m|C_{\max}$ considéré dans ce travail, est un cas particulier de $D_m||C_{\max}$ où, pour chaque job, une seule opération du deuxième étage a un temps opératoire non nul. Dès lors, les propriétés prouvées dans le cas du problème $A_m||C_{\max}$ restent valable pour le problème $F_{m+1}|T = m|C_{\max}$ [1,2].

À notre connaissance, le problème $F_{m+1}|T = m|C_{\max}$ pour des valeurs de $m > 2$ n'a pas été assez considéré dans la littérature [1,2], bien que nous rencontrons ce type de problème dans l'industrie. Dans ce contexte, nous proposons une nouvelle heuristique pour le problème $F_{m+1}|T = m|C_{\max}$.

Dans la section 2, nous présentons les différentes notations et les hypothèses utilisées. La section 3 expose certaines propriétés de base. Dans la section 4, nous proposons plusieurs bornes inférieures. La meilleure de ces bornes est utilisée dans l'heuristique développée dans la section 5. La section 6, présente une étude expérimentale, qui nous permet de comparer notre heuristique à celle proposée dans [9] et d'évaluer la qualité des solutions obtenues.

2. Notations et hypothèses

Les principales notations utilisées dans ce travail sont présentées dans la suite.

J	Ensemble des indices des jobs : $J = \{1, 2, \ldots, n\}$;
J_i	Job i, $i \in \{1, \ldots, n\}$;
S_k	Ensemble des indices des jobs de type k, $k \in \{1, \ldots, m\}$;
M_c	L'unique machine du premier étage (la machine commune) ;
M_k	La machine k, dédiée aux jobs de type k, $k \in \{1, \ldots, m\}$;
a_i	Temps opératoire du job i sur M_c ;
b_i	Temps opératoire du job i sur M_k, où $i \in S_k$;
$C_i(\pi)$	Date d'achèvement du job i sur M_c suivant la séquence π ;
$C_i^k(\pi)$	Date d'achèvement du job i sur M_k, $k \in \{1, \ldots, m\}$ suivant π ;
$C_{\max}^k(\pi)$	Date d'achèvement du dernier job de la séquence π sur M_k ;
$C_{\max}(\pi)$	Makespan de la séquence π ;
$C_{\max}$	Valeur minimale du makespan.

Un ensemble de n jobs composé chacun de deux opérations doit être réalisé sur deux étages. Le premier étage contient une seule machine M_c, commune à tous les jobs ; quant au deuxième, il contient m machines dédiées M_k, $k \in \{1, \ldots, m\}$. Chaque job de type k doit visiter d'abord la machine M_c, puis la machine M_k. Les n jobs sont tous disponibles à la date 0. Une machine donnée ne peut exécuter plus d'une opération à la fois, et un job donné ne peut être exécuté sur plus d'une machine à la fois. La préemption est interdite, et les jobs peuvent attendre entre les deux étages.

Le problème consiste à déterminer la séquence de jobs sur les machines qui minimise le makespan. Notons que pour une séquence π donnée, le makespan $C_{\max}(\pi)$ correspond au maximum entre les dates d'achèvement des derniers jobs visitant les machines M_k, $k \in \{1, \ldots, m\}$: $C_{\max}(\pi) = \max_{1 \leq k \leq m}(C^k_{\max}(\pi))$.

3. Propriétés de base

La propriété 3.1 permet de répondre à une question qui se pose en ordonnancement, et plus précisément dans le cas de problèmes NP-difficiles, sur la possibilité de limiter la recherche d'une solution optimale, aux ordonnancements de permutation.

Propriété 3.1. Pour le problème $F_{m+1}|T = m|C_{\max}$, les ordonnancements de permutation sont dominants.

Démonstration. La recherche d'une solution optimale au problème $A_m||C_{\max}$, peut se limiter aux ordonnancements de permutation [10]. Le lemme 1.1 permet d'étendre ce résultat de dominance des permutations, au problème miroir $D_m||C_{\max}$. Le problème $F_{m+1}|T = m|C_{\max}$ étant un cas particulier du problème $D_m||C_{\max}$, la propriété 3.1 est alors vérifiée. □

A partir d'ici, nous entendons par solution, uniquement les ordonnancements de permutation. Dans de tels ordonnancements, les jobs d'un même type k, visitent la machine M_k dans l'ordre de leur sortie de la machine M_c. Ainsi, la seule décision à prendre est celle concernant l'ordre suivant lequel, les jobs vont être exécutés sur M_c. Notons $a_{\min} = \min_{1 \leq i \leq n}(a_i)$ et $b_{\min} = \min_{1 \leq i \leq n}(b_i)$.

Nous avons alors la propriété suivante :

Propriété 3.2. Si pour un job donné i_0, nous avons :

$$b_{i_0} = \min\{a_{\min}, b_{\min}\}. \tag{3.1}$$

Alors il existe une solution optimale où le job i_0 est ordonnancé en dernière position.

Démonstration. Soit π une solution optimale, dans laquelle J_{i_0} n'est pas à la dernière position, avec $i_0 \in S_{k_0}$. Notons X, l'ensemble des jobs qui suivent J_{i_0} sur M_{k_0}, et J_{i_1} le dernier job de X (Fig. 1).

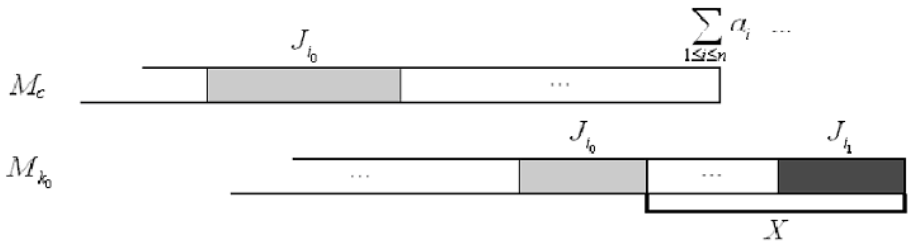

FIGURE 1. Ordonnancement ne respectant pas la propriété 3.2.

Soit π' la solution obtenue à partir de π, en plaçant J_{i_0} à la dernière position. On a, pour tout job $i \mid i \neq i_0$, $C_i(\pi') \leq C_i(\pi)$. D'où :

$$\forall k \neq k_0,\ C^k_{\max}(\pi') \leq C^k_{\max}(\pi). \tag{3.2}$$

D'après (3.1), les dates d'achèvement de tous les jobs de X, sur M_{k_0}, seront réduites d'au moins b_{i_0}, ainsi, $C_i^{k_0}(\pi') \leq C_i^{k_0}(\pi) - b_{i_0}$, $\forall i \in X$.

En particulier pour le job i_1 :

$$C_{i_1}^{k_0}(\pi') + b_{i_0} \leq C_{\max}(\pi). \tag{3.3}$$

En plaçant J_{i_0} à la dernière position sur M_c, nous obtenons :

$$C^{k_0}_{\max}(\pi') = C^{k_0}_{i_0}(\pi') = \max\left\{\sum_{1\leq i\leq n} a_i; C^{k_0}_{i_1}(\pi')\right\} + b_{i_0}. \tag{3.4}$$

Avec (3.1) et (3.3), (3.4) donne :

$$C^{k_0}_{\max}(\pi') \leq C_{\max}(\pi) \tag{3.5}$$

(3.2) et (3.5) permettent d'écrire : $C_{\max}(\pi') \leq C_{\max}(\pi)$. π' est alors optimale. □

Remarque 3.1. L'utilisation répétitive de la propriété 3.2, valable dans le cas $F_2||C_{\max}$, ne l'est plus dans le cas de machines dédiées. En effet, considérons l'exemple des 3 jobs suivants, où J_1 et J_2 sont de type 1 et J_3 est de type 2. Les valeurs respectives de (a_i, b_i) sont (2, 2), (3, 3) et (4, 4). L'application répétée de la propriété 3.2 donne la séquence (J_3, J_2, J_1) avec $C_{\max} = 12$, alors que la solution optimale est (J_2, J_3, J_1) avec $C_{\max} = 11$.

4. BORNES INFÉRIEURES

Nous développons dans ce qui suit, des bornes inférieures de la valeur du makespan, pour le problème $F_{m+1}|T = m|C_{\max}$.

Notons BI_0, la borne inférieure évidente donnée par :

$$BI_0 = \max\left(\sum_{i\in J} a_i + \min_{i\in J} b_i \ ; \ \max_{i\in J}(a_i + b_i)\right). \tag{4.1}$$

Par ailleurs, en considérant chaque type $k \in \{1, \ldots, m\}$ séparément, nous avons :

$$\min_{i\in S_k} a_i + \sum_{i\in S_k} b_i \leq C_{\max}. \tag{4.2}$$

Nous obtenons alors, la borne BI_1 donnée par :

$$BI_1 = \max_{1\leq k\leq m}\left(\min_{i\in S_k} a_i + \sum_{i\in S_k} b_i\right). \tag{4.3}$$

D'autre part, pour deux machines dédiées quelconques M_k et $M_{k'}$, avec $k \neq k'$, l'une de ces deux machines commence au plus tôt, après la sortie du deuxième job de M_c. Une borne inférieure du makespan est alors donnée, pour chaque paire de machines, par :

$$BI_2 = \max_{k\neq k'}\left(\min_{i\in S_k} a_i + \min_{i\in S_{k'}} a_i + \min(\sum_{i\in S_k} b_i, \sum_{i\in S_{k'}} b_i)\right). \tag{4.4}$$

Ce même raisonnement peut se faire en considérant p machines dédiées quelconques, l'une de ces machines commence au plus tôt, après la sortie de p jobs de M_c.

En prenant $p = m$, et en tenant compte de la machine dédiée sur laquelle la fabrication démarre en dernier, nous obtenons :

$$BI_m = \sum_{1\leq k\leq m}(\min_{i\in S_k} a_i) + \min_{1\leq k\leq m}\left(\sum_{i\in S_k} b_i\right). \tag{4.5}$$

Il serait alors intéressant de calculer la borne inférieure :

$$BI_{\max} = \max(BI_1, BI_2, \ldots, BI_m).$$

La valeur de $BI_{\max}$ est obtenue, en tenant compte pour chaque machine M_k, du temps nécessaire à la réalisation des jobs, et du temps d'attente minimal avant de démarrer la fabrication. D'où, $BI_{\max}$ correspond au makespan du problème particulier, composé d'un seul job par type, et où, les temps opératoires du job de type k sont : $\min_{i\in S_k} a_i$ sur M_c et $\sum_{i\in S_k} b_i$ sur M_k. La solution optimale de ce problème simple, s'obtient en suivant l'ordre décroissant des temps opératoires sur les machines dédiées. En effet, l'ordre inverse donne une solution optimale évidente, pour le problème où les machines dédiées sont au premier étage.

Une autre borne inférieure que nous proposons correspond à une généralisation de celle développée dans [9] pour le problème $F_3|T = 2|C_{\max}$ où les deux machines

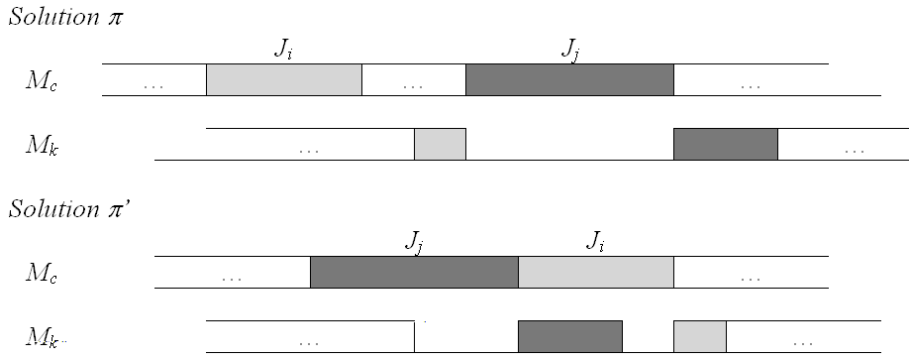

FIGURE 2. Inversion de l'ordre des jobs i et j.

dédiées sont au premier étage et la machine commune est au second. A partir du problème initial, nous définissons pour chaque type k, $k \in \{1, \ldots, m\}$; un problème de flow shop à deux machines, pour lequel l'algorithme de Johnson donne une permutation optimale π_k et un makespan $C^{Joh}_{\max}(\pi_k)$. Nous avons alors :

$$BI_{joh} = \max_{1 \leq k \leq m} \{C^{Joh}_{\max}(\pi_k)\}. \tag{4.6}$$

Finalement, nous pouvons utiliser la meilleure de toutes ces bornes en prenant :

$$BI = \max(BI_0, BI_{\max}, BI_{joh}). \tag{4.7}$$

5. CAS POLYNOMIAUX

Chacun des deux lemmes suivants correspond à un cas particulier du problème $F_{m+1}|T = m|C_{\max}$ pouvant être résolu en un temps polynomial.

Lemme 5.1. *Si pour tout job i, on a $b_i \leq a_i$, alors le problème $F_{m+1}|T = m|C_{\max}$ est de complexité polynomiale.*

Démonstration. Vérifions d'abord que sur chaque machine dédiée, l'ordre décroissant des b_i est un ordre optimal.

Soit π une permutation optimale ne respectant pas cet ordre. Il existe alors, deux jobs i et j de même type k, avec J_i placé avant J_j, et $b_i < b_j$. Considérons π', la permutation obtenue à partir de π, en insérant le job i à la suite du job j (voir Fig. 2).

Ainsi, sur M_c, pour tous les jobs $i' \neq i$, soit ils restent inchangés, soit ils avancent de a_i. D'où, pour toutes les machines $M_{k'} \neq M_k$, nous avons : $C^{k'}_{\max}(\pi') \leq C^{k'}_{\max}(\pi)$.

Quant au job j et aux jobs de type k placés entre J_i et J_j, ils avancent de a_i sur M_c. Comme $b_i \leq a_i$, ces jobs avancent d'au moins b_i sur M_k.

En plaçant J_i à la suite de J_j, nous avons : $C_i^k(\pi') = \max(C_i(\pi'), C_j^k(\pi')) + b_i$. Or, $C_i(\pi') = C_j(\pi)$ et $C_j^k(\pi') \leq C_j^k(\pi) - b_i$. D'où $C_i^k(\pi') \leq C_j^k(\pi)$.

Ce qui permet d'écrire pour tous les jobs i' suivant J_i sur M_k : $C_{i'}^k(\pi') \leq C_{i'}^k(\pi)$. Et par conséquent : $C_{\max}^k(\pi') \leq C_{\max}^k(\pi)$.

D'où $C_{\max}(\pi') \leq C_{\max}(\pi)$, et l'ordre décroissant des b_i est alors un ordre optimal sur les machines dédiées.

Afin de déterminer l'ordre optimal sur M_c, considérons le problème miroir dans lequel les machines dédiées sont situées au premier étage et la machine commune au second. Pour ce problème, le lemme 1.1. permet d'affirmer que l'ordre croissant des b_i est un ordre optimal sur les machines M_k. Minimiser le makespan revient à minimiser les temps d'attente sur la machine M_c du second étage, et donc à prendre les jobs suivant l'ordre selon lequel ils sortent des machines M_k.

En inversant l'ordre de cette permutation, nous obtenons une permutation optimale de notre problème. □

Pour les besoins du lemme suivant, considérons pour chaque type k, la permutation π_k donnant les jobs de S_k, dans l'ordre croissant des a_i. On peut supposer, sans perte de généralités, que nous disposons du même nombre n_0, de jobs par type . En effet, il suffit d'ajouter des jobs fictifs, avec deux opérations de durées nulles, jusqu'à atteindre pour chaque type, un nombre de jobs $n_0 = \max_k n_k$.

Considérons maintenant, la permutation π qui prend d'abord les m premiers jobs de chaque type (suivant un ordre arbitraire), ensuite les m seconds .... Cette permutation permet d'alterner les jobs de différents types, en commençant par les machines dédiées ayant le plus grand nombre de jobs. Dans un même m-uplet, les jobs (non fictifs) appartiennent à des machines dédiées qui ont le même nombre de jobs non encore placés. Cette permutation peut être donnée par :

$$\pi(jm + k) = \pi_k(j + 1),\ k = 1, \ldots, m \text{ et } j =, \ldots, n_0 - 1. \tag{5.1}$$

Considérons l'exemple suivant avec $m = 3$ et $n = 22$, avec J_1, .., J_{10} de type 1 ; J_{11}, .., J_{17} de type 2 ; et J_{18}, .., J_{22} de type 3 . Supposons que les jobs sont donnés pour chaque type dans l'ordre croissant des a_i. Nous obtenons $n_0 = 10$, avec 3 jobs fictifs placés en début de la liste des jobs de type 2 et, 5 jobs fictifs en début de la liste des jobs de type 3. La permutation π est donnée par : ((1, 0, 0), (2, 0, 0), (3, 0, 0), (4, 11, 0), (5, 12, 0), (6, 13, 18), (7, 14, 19), (8, 15, 20), (9, 16, 21), (10, 17, 22)).

Lemme 5.2. *Si la durée de la deuxième opération est constante, $b_i = b$ pour tout job i (non fictif), alors la permutation π, donnée dans (5.1), est optimale.*

Démonstration. Dans ce qui suit, nous ne considérons que les jobs non fictifs.

Comme $b_i = b$ pour tout job i, les jobs d'un même m-uplet appartiennent à des machines dédiées ayant la même charge restante. D'où, si nous considérons une fabrication au plus tard, ces jobs sont tous réalisés à la même date. Ainsi, pour J_i de type k et appartenant au $j^{\text{ème}}$ m-uplet, si J_i est le job définissant la valeur du $C_{\max}$ alors il ne peut être que le dernier job du m-uplet auquel il appartient.

Nous avons alors :

$$C_{\max}(\pi) = \sum_{p=1}^{(j-1)m+k} a_{\pi(p)} + (n_0 - j + 1)b.$$

Si J_i appartient au dernier m-uplet ($j = n_0$) alors : $C_{\max}(\pi) = \sum_{p=1}^{n_0 m} a_{\pi(p)} + b$.

Cette valeur est optimale car elle correspond à une borne inférieure du $C_{\max}$.

Sinon, $j < n_0$, alors la durée totale sur M_c du $(j+1)^{\text{ème}}$ m-uplet est inférieure ou égale à b. Comme pour chaque type l'ordre de placement est celui des a_i croissants, chaque m-uplet du 1^{er} au $j^{\text{ème}}$ possède alors une durée totale sur M_c, inférieure ou égale à b. D'où, le job i appartient au 1^{er} m-uplet (J_i correspond à l'une des machines dédiées les plus chargées).

On a ainsi : $C_{\max} = \sum_{k \in I} \min_{i \in S_k} a_i + n_0 b$.

Cette valeur est optimale car elle correspond aussi, à une borne inférieure du makespan. □

6. Heuristique proposée

Avant de décrire l'heuristique H_{DHH} que nous proposons, nous commençons par généraliser l'heuristique présentée dans [9], à un nombre m quelconque de machines dédiées. Nous la notons H_{OLC}.

Heuristique H_{OLC}

Etape 1

Pour tout $k \in \{1, \ldots, m\}$, appliquer l'algorithme de Johnson à l'atelier (M_k, M_c) et aux jobs de S_k, soit π^k_{joh} la permutation optimale obtenue.

Etape 2

Placer les jobs sur les différentes machines du premier étage conformément à leur ordre d'apparition dans les séquences π^k_{joh}, $k \in \{1, \ldots, m\}$. Et calculer les dates d'achèvement C^k_i des jobs $i \in \{1, \ldots, n\}$ sur M_k, $k \in \{1, \ldots, m\}$.

Etape 3

La séquence π_{OLC} obtenue en suivant pour les n jobs l'ordre décroissant des C^k_i, $i \in \{1, \ldots, n\}$ et $k \in \{1, \ldots, m\}$, est solution du problème où les machines dédiées sont au second étage.

Cette heuristique, de complexité $O(n \log n)$, a été proposée pour le cas $m = 2$. Le raisonnement adopté, basée sur l'algorithme de Johnson, peut pour certains jeux de données, ne pas être satisfaisant. Afin d'illustrer cela, considérons une instance de 8 jobs dans laquelle les 4 premiers jobs sont de type 1 et les 4 derniers de type 2. Les durées opératoires sont données dans le tableau 1.

En appliquant H_{OLC}, nous obtenons l'ordre 1, 5, 2, 6, 3, 7, 4, 8 ; avec un makespan égal à 37. Remarquons que l'ordre inverse 8, 4, 7, 3, 6, 2, 5, 1 correspond

TABLEAU 1. Temps opératoires.

	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
a_i	2	3	4	5	2	3	4	5
b_i	3	4	5	9	3	4	5	9

à une solution optimale, avec un makespan égal à 31, cette valeur représentant une borne inférieure.

L'heuristique H_{OLC} peut alors être, bien adaptée et fournir de bonnes solutions, pour certaines familles d'instances plutôt que pour d'autres. Ceci nous conduit à proposer une nouvelle heuristique (H_{DHH}), qui permettrait de remédier à cette défaillance.

Dans l'heuristique H_{DHH} que nous proposons, nous construisons une séquence, en remontant de la dernière position à la première. Nous cherchons, par le choix du job à placer à une position donnée, à ne pas dépasser la borne inférieure BI. Dans le cas où un tel job n'existe pas, le job retenu est celui qui minimise le dépassement de BI.

Les notations suivantes sont utiles pour détailler la description de l'heuristique H_{DHH} :

- π Solution construite par H_{DHH} ;
- J^p Ensemble des jobs ordonnancés à l'itération $n-p$;
- $\bar{J}^p$ Ensemble des jobs non encore ordonnancés à l'itération $n-p$;
- D_c Date de fin des jobs de $\bar{J}^p$ sur M_c ;
- D_k Date de fin souhaitée, pour le prochain placement sur M_k, $k \in \{1,\ldots,m\}$;
- X_k Ensemble de jobs de type k, candidats au placement, $k \in \{1,\ldots,m\}$.

À la première itération, nous choisissons le job à placer à la position n, sans dépasser la borne inférieure BI. A la fin de l'itération $n-p$ ($p = n,\ldots,1$), les p derniers jobs de π sont fixés (J^p). Pour qu'un job de type k dans $\bar{J}^p$, soit candidat au placement, il faut qu'il ait la possibilité de finir sur M_k, à une date ne dépassant pas D_k. Dans le cas où plusieurs candidats existent, plusieurs règles de choix peuvent être utilisées. Si aucun job ne permet de maintenir cette date, nous choisissons dans $\bar{J}^p$, celui qui minimise le dépassement de BI.

Au début de la procédure, si la condition (3.1) de la propriété 3.2 est vérifiée, la taille du problème est réduite d'un job, celui-ci étant placé à la fin de la séquence.

Algorithme de l'heuristique H_{DHH}

Initialisation

$p = n$; $\bar{J}^n = J$; $J^n = \emptyset$; $D_c = \sum_{i=1}^{n} a_i$ $D_k = BI$, $k \in \{1,\ldots,m\}$; $\pi(i) = 0$, pour $i = 1,\ldots,n$; $C_{\max}(\pi) = BI$;

Etape 1 : Vérification de la propriété 3.2

Chercher un job i_0 tel que $b_{i_0} = \min_{\bar{J}^p}(a_i, b_i)$
Si un tel job n'existe pas aller à l'étape 2.

Ordonnancer J_{i_0} à la $p^{\text{ème}}$ position : $\pi(p) = i_0$; $J^{p-1} = J^p \bigcup \{J_{i_0}\}$;
$\bar{J}^{p-1} = \bar{J}^p \backslash \{J_{i_0}\}$;
Pour $i_0 \in S_{k_0}$, faire : $D_{k_0} = D_{k_0} - b_{i_0}$; $D_c = D_c - a_{i_0}$; $p = p - 1$.

Etape 2 : Construction de la séquence

Pour $k \in \{1, \ldots, m\}$: $X_k = \{J_i \in \bar{J}^p \bigcap S_k | b_i \leq D_k - D_c\}$, posons $X^p = \bigcup_{1 \leq k \leq m} X_k$.
Si $X^p \neq \emptyset$

Choisir dans X^p, le job i_0 de type k_0.
$D_{k_0} = D_{k_0} - b_{i_0}$.

Sinon

Soit $J_{i_0} \in \bar{J}^p | b_{i_0} - (D_{k_0} - D_c) = \min_{i \in \bar{J}^p | i \in S_k} \{b_i - (D_k - D_c)\}$; où $J_{i_0} \in S_{k_0}$.

Calculer $\Delta = b_{i_0} - (D_{k_0} - D_c)$;
$C_{\max}(\pi) = C_{\max}(\pi) + \Delta$; $D_{k_0} = D_c$;
$D_{k_1} = D_{k_1} + Delta$ pour $k_1 \neq k_0,\ k_1 \in \{1, \ldots, m\}$;

Fin sinon

Ordonnancer J_{i_0} à la $p^{\text{ème}}$ position : $\pi(p) = i_0$; $J^{p-1} = J^p \bigcup \{J_{i_0}\}$;
$\bar{J}^{p-1} = \bar{J}^p \backslash \{J_{i_0}\}$; $D_c = D_c - a_{i_0}$; $p = p - 1$.

Etape 3

Si $p \neq 0$ aller à l'étape 2 ;
Sinon : Fin de l'algorithme.

À l'étape 1, nous cherchons le minimum parmi $2n$ éléments, la complexité de cette partie est alors en $O(n)$. À chaque itération de l'étape 2, la sélection du job à placer nécessite $O(n)$. Ensuite, m opérations sont effectuées pour mettre à jour les valeurs D_i (avec $m \leq n$). L'étape 2 peut être répétée au plus n fois. La deuxième étape est donc en $O(n^2)$. Ainsi l'heuristique H_{DHH} est en $O(n^2)$.

À l'étape 2, lorsque plusieurs candidats au placement existent, le choix dans X^p du job à placer peut se faire soit aléatoirement, ou encore en fixant des règles de choix telles que :

- Calculer $\alpha_i = D_k - D_c + a_i - b_i$ pour tout $J_i \in X_k$ ($k \in \{1, \ldots, m\}$). Choisir $J_{i_0} \in X^p | \alpha_{i_0} = \max_{i \in \bigcup_{1 \leq k \leq m} X_k} \{\alpha_i\}$.
- Placer le job i ayant le plus grand rapport a_i/b_i.

Nous cherchons par ces deux règles à augmenter les possibilités de placement à l'itération suivante.

Dans la première règle, si à une itération donnée un job i de type k est placé, α_i représente l'espace disponible au prochain placement sur M_k. Dans cette règle, on favorise le placement sur la machine M_k, qui jusque là, reste la moins sollicitée (D_k grand).

Dans la deuxième règle, nous cherchons parmi les candidats au placement, et indépendamment de l'occupation des machines dédiées, le job correspondant au plus grand rapport a_i/b_i.

Remarquons que, dans l'étape 2, si à chaque itération X^p est non vide, l'algorithme H_{DHH} fournit une solution optimale. En effet, dans ce cas, la valeur du makespan est égale à la borne inférieure.

7. Résultats expérimentaux

Dans cette partie, nous commençons d'abord par décrire les classes d'instances que nous allons utiliser pour tester et comparer les deux heuristiques. Par la suite, nous présentons et analysons les différents résultats obtenus.

7.1. Génération des problèmes tests

Afin de mesurer les performances de l'heuristique H_{DHH}, nous avons effectué une étude comparative avec l'heuristique H_{OLC}. Cette étude expérimentale est réalisée sur plusieurs classes d'instances, $Cl1$ à $Cl7$, générées aléatoirement. Chaque classe contient 24 tailles différentes ($n = 40, 60, 80, \ldots, 500$). Pour chaque taille, 50 instances sont générées et les jobs sont, tant que possible, équitablement divisés entre les différents types de jobs. Les valeurs de m varient de 2 à 6.

Les temps opératoires des différents jobs suivent une distribution uniforme. Dans la classe $Cl1$, ces temps sont générés dans [1,20]. Pour $Cl2$, les temps opératoires sur la machine M_c sont générés dans [1,20] et, ceux des machines dédiées dans [1,40]. Ainsi, pour $Cl1$, les temps opératoires ont le même ordre de grandeur, quant à $Cl2$, les temps opératoires sur les machines dédiées sont plus importants que ceux sur la machine M_c.

Dans les classes $Cl3$ à $Cl7$, nous avons cherché à étudier des classes particulières qui pourraient se prêter plus au raisonnement proposé dans l'un ou l'autre des deux algorithmes. Dans la classe $Cl3$, pour tout job i, la deuxième opération nécessite plus de temps que la première. Pour cela nous avons généré des instances dans [1,20], avec $a_i \leq b_i$. Pour les classes $Cl4$ et $Cl5$, nous supposons que, pour deux jobs donnés i et j, si $a_i < a_j$ alors $b_i < b_j$. Pour cela nous avons introduit un paramètre ϵ, et généré les temps opératoires de la manière suivante :
Étape 1 : générer a_1 et b_1 dans [1,20] tel que $a_1 < b_1$;
Étape 2 : pour $i = 2$ à n, générer a_i dans $[a_{i-1}, a_{i-1} + \epsilon]$ et b_i dans $[\max(a_i, b_{i-1}), \max(a_i, b_{i-1}) + \epsilon]$.

$Cl4$ (resp. $Cl5$) correspond à $\epsilon = 5$ (resp. $\epsilon = 10$).
Enfin, les classes $Cl6$ et $Cl7$ sont telles que a_i soit générée dans [1,20] et $b_i = a_i + \gamma$, où $\gamma = 5$ pour $Cl6$ et $\gamma = 10$ pour $Cl7$.

Pour chaque instance générée, nous calculons la valeur de la borne inférieure BI donnée dans (4.7). Nous avons aussi, testé les deux règles de choix de placement d'un job à l'étape 2 de l'heuristique H_{DHH}. Les résultats obtenus sont, en moyenne, équivalents.

TABLEAU 2. Valeurs moyennes et maximales des écarts de H_{DHH}.

	E^{moy}_{DHH}						
	$Cl1$	$Cl2$	$Cl3$	$Cl4$	$Cl5$	$Cl6$	$Cl7$
$m=2$	0,22	0,22	0,69	0,38	0,43	0,74	1,12
$m=3$	0,29	0,22	0,82	0,37	0,41	0,64	0,66
$m=4$	0,26	0,30	0,81	0,34	0,39	0,64	0,63
$m=5$	0,41	0,38	0,79	0,33	0,38	0,63	0,61
$m=6$	0,34	0,25	0,80	0,33	0,36	0,62	0,63
	$E^{\max}_{DHH}$						
	$Cl1$	$Cl2$	$Cl3$	$Cl4$	$Cl5$	$Cl6$	$Cl7$
$m=2$	1,19	2,16	5,23	4,82	5,88	5,20	7,51
$m=3$	0,88	0,85	6,84	4,91	5,03	4,82	4,58
$m=4$	0,96	1,66	5,91	5,29	4,68	4,80	4,24
$m=5$	0,79	1,70	6,48	4,81	4,93	5,26	4,88
$m=6$	1,64	0,68	6,05	4,04	4,66	4,71	4,56

TABLEAU 3. Valeurs moyennes et maximales des écarts de H_{OLC}[1].

	E^{moy}_{OLC}						
	$Cl1$	$Cl2$	$Cl3$	$Cl4$	$Cl5$	$Cl6$	$Cl7$
$m=2$	0,22	0,32	0,43	0,77	0,92	1,05	0,91
$m=3$	–	0,50	0,54	0,71	0,88	1,02	1,01
$m=4$	0,23	0,13	0,54	0,71	0,86	1,00	0,99
$m=5$	0,26	0,38	0,51	0,69	0,87	0,96	0,96
$m=6$	–	0,28	0,46	0,70	0,87	0,92	0,92
	$E^{\max}_{OLC}$						
	$Cl1$	$Cl2$	$Cl3$	$Cl4$	$Cl5$	$Cl6$	$Cl7$
$m=2$	0,22	0,42	5,09	7,04	7,38	5,84	4,80
$m=3$	–	0,50	6,47	7,47	6,39	5,00	4,58
$m=4$	0,25	0,17	5,73	6,33	6,43	5,16	4,66
$m=5$	0,26	0,80	5,42	6,09	5,86	4,30	4,19
$m=6$	–	0,56	5,88	5,68	6,87	4,02	4,09

7.2. ANALYSE DES RÉSULTATS

Pour chaque instance, nous calculons l'écart, en pourcentage, par rapport à la borne inférieure BI, obtenu par l'heuristique H_{DHH} : $E_{DHH} = (C_{\max}(\pi) - BI) \cdot 100/BI$. De façon similaire, nous calculons l'écart en pourcentage E_{OLC} obtenu par H_{OLC}.

Dans les tableaux 2 et 3, nous donnons les valeurs moyennes et maximales des écarts E_{DHH} et E_{OLC} que nous notons respectivement E^{moy}_{DHH}, $E^{\max}_{DHH}$, E^{moy}_{OLC} et $E^{\max}_{OLC}$. La moyenne des écarts est calculée sur les seules instances pour lesquelles la borne inférieure est dépassée.

TABLEAU 4. Pourcentage de solutions avec $C_{\max} = BI$.

	PR_{DHH}				
	$m = 2$	$m = 3$	$m = 4$	$m = 5$	$m = 6$
$Cl1$	98,00	98,42	98,17	98,83	98,67
$Cl2$	81,92	97,33	97,17	97,92	98,00
	PR_{OLC}				
	$m = 2$	$m = 3$	$m = 4$	$m = 5$	$m = 6$
$Cl1$	99,92	100,00	99,75	99,92	100,00
$Cl2$	99,83	99,92	99,75	99,75	99,67

Les tableaux 2 et 3 montrent que les classes $Cl1$ et $Cl2$ sont faciles à résoudre par l'une ou l'autre des deux heuristiques, avec une légère préférence pour H_{OLC}, et qu'une solution optimale est vite trouvée. Par contre, et mise à part le cas de la classe $Cl3$ et le cas $m = 2$ pour la classe $Cl7$, où H_{OLC} est la plus proche de la borne inférieure ; l'heuristique H_{DHH} est nettement meilleure que H_{OLC} pour le reste des classes.

Dans la classe $Cl7$, la durée de la deuxième opération est très importante par rapport à la première. Lorsqu'à une itération donnée, H_{DHH} ne trouve aucun candidat au placement, on choisit parmi les jobs i restants, celui ayant le plus faible b_i et, par conséquent, un a_i encore plus faible. Lorsque $m = 2$, et malgré l'alternance entre les deux machines dédiées, on se trouve fréquemment, incapable d'éviter d'autres retards pour les placements à suivre : en effet, souvent pour 2 jobs i et j, on a $2a_i \leq b_j$.

Dans la classe $Cl3$, malgré que $a_i \leq b_i$, l'appartenance de ces deux durées au même intervalle, fait qu'en général, M_c est la machine la plus chargée. Il est alors très probable que la borne inférieure soit donnée par BI_0. H_{DHH} aura alors tendance à placer, à la fin de la séquence les jobs i ayant des b_i faibles et donc des a_i faibles. Ce qui limite les choix dans les itérations à suivre, et on se trouvera à plusieurs reprises contraint à dépasser la borne inférieure.

Par contre et pour les deux classes $Cl7$ et $Cl3$, l'ordre pour chacun des types de jobs retenu par H_{OLC} est celui des a_i croissants. D'où, le dernier job placé correspond toujours au plus grand a_i et par conséquent à un b_i grand. Ce choix va induire généralement un dépassement de la borne inférieure, mais offre beaucoup d'aisance pour le placement des autres jobs.

Pour confirmer la facilité de résolution des classes $Cl1$ et $Cl2$, nous donnons dans le tableau 4, le pourcentage PR_{DHH} (resp. PR_{OLC}) d'instances, appartenant à ces classes, pour lesquelles l'heuristique H_{DHH} (resp. H_{OLC}) donne un makespan égal à la borne inférieure.

Il est intéressant de remarquer également, que les valeurs des écarts moyens obtenus par les deux heuristiques et dans tous les cas de figures, sont à moins de 1,12 % de l'optimum (Tab. 2 et 3) ; et au pire des cas à moins de 8 %.

[1](-) E^{moy}_{OLC} est non défini car toutes les solutions vérifient $C_{\max} = BI$.

TABLEAU 5. Comparaison relative des deux heuristiques.

	H_{DHH} meilleure que H_{OLC}						
	$Cl1$	$Cl2$	$Cl3$	$Cl4$	$Cl5$	$Cl6$	$Cl7$
$m=2$	0,0	0,0	1,67	53,58	66,83	92,67	28,67
$m=3$	0,0	0,0	2,50	47,08	63,83	92,75	94,25
$m=4$	0,0	0,0	1,83	47,50	61,58	92,75	92,00
$m=5$	0,0	0,0	1,50	46,83	60,92	90,67	90,58
$m=6$	0,0	0,0	1,33	48,58	62,17	89,83	88,83
	H_{OLC} meilleure que H_{DHH}						
	$Cl1$	$Cl2$	$Cl3$	$Cl4$	$Cl5$	$Cl6$	$Cl7$
$m=2$	2,00	17,92	76,17	19,92	11,67	1,08	64,17
$m=3$	1,58	2,58	90,92	24,50	10,17	1,25	0,75
$m=4$	1,75	2,75	91,75	21,83	10,67	1,83	1,58
$m=5$	1,08	1,92	91,92	16,92	9,25	3,50	2,92
$m=6$	1,33	1,75	92,50	15,08	9,00	5,00	5,50

Dans le tableau 5, nous donnons, pour chacune des deux heuristiques, le pourcentage d'instances pour lesquelles elle est strictement meilleure.

Les résultats du tableau 5 montrent que l'heuristique H_{OLC} devance l'heuristique H_{DHH} pour les classes $Cl1$ et $Cl2$. Ce résultat reste très relatif, vu le pourcentage très important de solutions optimales obtenues par les deux heuristiques (Tab. 4), et la valeur très faible des écarts moyens et maximaux (Tab. 2 et 3). En outre, ces résultats mettent en exergue le fait que les performances des deux heuristiques, peuvent être très différentes, et qu'elles sont plutôt complémentaires, et dépendent de la structure de données. Ainsi, pour la classe $Cl3$ et le cas $m=2$ pour la classe $Cl7$, l'heuristique H_{OLC} donne les meilleurs résultats, alors que pour le reste des classes, l'heuristique H_{DHH} permet d'obtenir de meilleurs résultats.

Il est à remarquer que les durées d'exécution des heuristiques peuvent être considérées comme négligeables. En effet, pour toutes les classes et toutes les instances, la somme des durées d'exécution des deux heuristiques reste inférieure à 14 min, avec un processeur de 1 Ghz (soit moins de 0,02 s par instance).

Enfin, la complémentarité des deux heuristiques ainsi que leur rapidité d'exécution, nous conduisent, logiquement, à utiliser les deux heuristiques et à retenir la meilleure des deux solutions et ce, pour tout type de données.

8. CONCLUSION

Dans ce papier, nous avons traité le problème de flow shop hybride à deux étages avec machines dédiées. Nous avons présenté certaines propriétés permettant de limiter le domaine de recherche. Le problème étant NP-difficile au sens fort, plusieurs bornes inférieures sont données et une nouvelle approche de résolution est développée. La solution est obtenue en plaçant à chaque itération, le dernier job de la séquence partielle et ce, en se basant sur la valeur de la meilleure borne inférieure

obtenue. Une étude comparative avec une heuristique de la littérature, que nous avons généralisée au cas $m > 2$, est proposée. Appliquée sur un très grand nombre d'instances réparties en différentes classes selon la nature des données, cette étude a montré une complémentarité de ces deux approches, avec une solution de très bonne qualité, qui ne s'écarte pas de la borne inférieure, de plus de 1,12 % en moyenne et de plus de 8 % au pire des cas.

L'étude du même type d'atelier en considérant d'autres critères et en intégrant d'autres types de contraintes représente à notre avis une suite intéressante à ce travail.

Références

[1] H. Hadda, N. Dridi and S. Hajri-Gabouj, Heuristiques pour le flow shop hybride à deux étages avec machines dédiées, in *Actes de la Conférence Scientifique Conjointe en Recherche Opérationnelle et aide à la Décision : FRANCORO V/ROADEF*. Grenoble, France (2007) 229–230.

[2] H. Hadda, N. Dridi and S. Hajri-Gabouj, Heuristique d'ordonnancement du flow shop hybride à deux étages avec machines dédiées, in *Actes du 7ème Congrès International de Génie Industriel : CIGI*. Trois-Rivières, Québec (2007).

[3] M. Haouari and T. Daouas, Optimal scheduling of the 3-machine assembly-type flow shop. *RAIRO Oper. Res.* **33** (1999) 439–445.

[4] S.M. Johnson, Optimal two- and three-stage production schedules with setup times included. *Nav. Res. Log. Q.* **1** (1954) 61–68.

[5] C.Y. Lee, T.C.E. Cheng and B.M.T. Lin, Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem. *Manage. Sci.* **39** (1993) 616–625.

[6] B.M.T. Lin, The strong NP-hardness of two-stage flowshop scheduling with a common second-stage machine. *Comput. Oper. Res.* **26** (1999) 695–698.

[7] H.T. Lin and C.J. Liao, A case study in a two-stage hybrid flow shop with setup time and dedicated machines. *Int. J. Prod. Econ.* **86** (2003) 133–143.

[8] M. Lin and J. Wu, Effective lower bounds for scheduling problems in two-stage hybrid flowshops. *J. Manage.* **21** (2004) 363–374.

[9] C. Oguz, B.M.T. Lin and T.C.E. Cheng, Two-stage flowshop scheduling problem with a common second-stage machine. *Comput. Oper. Res.* **24** (1997) 1169–1174.

[10] C.N. Potts, S.V. Sevast'janov, V.A. Strusevich, L.N. Van, wassenhove and C.M. Zwaneveld, The two-stage assembly scheduling problem : complexity and approximation. *Oper. Res.* **43** (1995) 346–355.

[11] F. Riane, A. Artiba and S.E. Elmaghraby, Sequencing hybrid two-stage flow shop with dedicated machines, in *3e Conférence Francophone de modélisation et simulation « Conception, Analyse et Gestion des Systèmes Industriels », MOSIM'01*, Troyes, France (2001) 591–597.

[12] F. Riane, A. Artiba and S.E. Elmaghraby, A hybrid three-stage flowshop problem: Efficient heuristics to minimise makespan. *Eur. J. Oper. Res.* **109** (1998) 321–329.