HAL
open science

NP-complétude du problème du flowshop à deux machines avec des opérations couplées

Nadjat Meziani, Ammar Oulamara, Mourad Boudhar

To cite this version:

Nadjat Meziani, Ammar Oulamara, Mourad Boudhar. NP-complétude du problème du flowshop à deux machines avec des opérations couplées. Xème Conférence Internationale : Conception et Production Intégrées, Dec 2015, Tanger, Maroc. hal-01260752

HAL Id: hal-01260752

https://hal.science/hal-01260752v1

Submitted on 22 Jan 2016

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.

NP-complétude du problème du flowshop à deux machines avec des opérations couplées

Nadjat Meziani
Faculté des Sciences
Economiques, Commerciales
et des Sciences de Gestion,
Université Abderrahmane
Mira Béjaia, Algérie
Email : ro_nadjet07@yahoo.fr

Ammar Oulamara
LORIA-UMR 7503,
Université de Lorraine,
Campus Scientifique-BP 239,
54506 Vandoeuvre-les-Nancy
Cedex, France
Email : Ammar.Oulamara@loria.fr

Mourad Boudhar
Laboratoire RECITS,
Faculté de Mathématiques,
USTHB, BP 32,
Bab-Ezzouar, El-Alia 16111,
Alger, Algérie
Email : mboudhar@yahoo.fr

***Résumé*—Dans ce travail, Nous considérons le problème du flowshop à deux machines avec des opérations couplées. Chaque tâche est composée de deux opérations couplées sur la première machine séparées par un délai exact et d'une seule opération sur la deuxième machine. L'objectif est de minimiser le makespan. Nous étudions la complexité d'un sous problème et nous montrons qu'il est NP-difficile. Pour la résolution du problème général, nous proposons des heuristiques avec des expérimentations numériques et nous présentons des sous problèmes polynomiaux.**

Mots-clés : flowshop, opérations-couplées, délai exact, makespan.

I. Introduction

Le problème d'ordonnancement de tâches couplées a été introduit pour la première fois par Shapiro [15]. Le problème consiste à ordonnancer un ensemble de tâches sur une seule machine. Chaque tâche est constituée de deux opérations, séparées par un délai exact, qui s'exécutent dans l'ordre. L'objectif est de minimiser la date de fin de traitement des tâches (makespan). La motivation de ce problème découle d'un problème d'ordonnancement des tâches d'un radar, qui consiste dans l'émission des impulsions et à la réception des réponses après un temps d'attente. Orman et Potts [14] ont étudié le problème de tâches couplées sur une seule machine pour minimiser le makespan. Ils ont dénombré plusieurs problèmes et les ont hiérarchisés selon leurs complexités. Ahr et al [3] proposent un algorithme exact utilisant la programmation dynamique qui permet de résoudre le problème pour de petites instances où L est fixé. Cet algorithme a été adapté par Brauner et al dans [5] pour résoudre un problème de tâches couplées motivé par les problèmes de gestion du temps de production cyclique avec des robots. Autres chercheurs se sont dirigés vers l'approximation de ces problèmes. Ainsi Ageev et Baburin [1] proposent une 7/4 et 3/2-approximation pour résoudre les problèmes $1/$*tâche-couplée*$, a_i = b_i = 1, L_i/C_{max}$ et $F2/$*tâche-couplée*$, a_i = b_i = 1, L_i/C_{max}$, respectivement. Pour les problèmes $1/$*tâche-couplée*$, a_i, L_i, b_i/C_{max}$ et $F2/$*tâche-couplée*$, a_i, L_i, b_i/C_{max}$, Ageev et Kononov [2] donnent plusieurs résultats d'approximation et des bornes de non approximabilité selon les valeurs de a_i et b_i. Peu de travaux ont été réalisés en rajoutant des contraintes aux tâches couplées. Blazewicz et al [4] ont prouvé que le problème polynomial $1/$*tâche-couplée*$, a_i = b_i = 1, L_i = l/C_{max}$ est NP-complet en ajoutant une contrainte de précédence entre les tâches couplées. Yu et al [17] ont prouvé que le problème à deux machines $F2/$*tâche-couplée*$, a_i = b_i = 1, L_i/C_{max}$ est NP-difficile ce qui découle que le problème à une machine $1/$*tâche-couplée*$, a_i = b_i = 1, L_i/C_{max}$ est aussi NP-difficile [17]. Simonin et al ont étudié dans [16] le problème de tâches couplées avec des tâches de traitement et d'acquisition en présence des contraintes de précédence et d'incompatibilité. Ils ont proposé un algorithme polynomial pour ce spécial problème. Dans [12], les auteurs ont abordé le problème $F2/Coup - Oper(1), a_i = L_i = p, b_i, c_i/C_{max}$ et ils ont montré qu'il est NP-difficile. Ils ont exposé également des sous problèmes polynomiaux. En ordonnancement, le problème de tâches couplées avec un délai exact est un cas particulier du problème de time lag dans lequel le délai correspond à un délai minimum et maximum de même valeur. Mitten a fourni dans [13] un algorithme polynomial pour minimiser le makespan pour le problème du flowshop de permutation à deux machines avec un délai minimum. D'autres travaux sont abordés dans ce contexte par certains auteurs dans [6][7][11].

II. Description du problème

Dans ce travail, nous considérons le problème du flowshop à deux machines où $J = \{1, \ldots, n\}$ est l'ensemble de n tâches à traiter sur les deux machines. Chaque tâche est composée d'un couple d'opérations $O_{1,j}$ sur la première machine et d'une seule opération $O_{2,j}$ sur la deuxième machine. Chaque couple d'opérations de tâches $O_{1,j}$ est noté par le triplet (a_j, L_j, b_j), telles que a_j et b_j sont les durées de traitement de la première et de la deuxième opération du couple d'opération

Xème Conférence Internationale : Conception et Production Intégrées, CPI 2015, 2-4 Décembre 2015, Tanger - Maroc.
Xth International Conference on Integrated Design and Production, CPI 2015, December 2-4, 2015, Tangier - Morocco.

$O_{1,j}$, respectivement. L_i est le délai exact qui s'écoule entre les dates de fin et début de traitement de la première et de la deuxième opération, respectivement. La valeur c_j est le temps d'exécution de l'opération $O_{2,j}$ qui ne peut être traitée que si le traitement du couple d'opérations $O_{1,j}$ est terminé. Notre objectif est de minimiser la date de fin de traitement des tâches (makespan). Cependant, suivant la notation introduite par Graham et al [9], notre problème est noté par $F2/Coup_Opr(1), a_i, b_i, L_i, c_i/C_{max}$.

III. Résultats de complexité

Dans ce papier, Nous avons établi quelques nouveaux résultats en rapport avec le problème décrit ci-dessus, et que nous énonçons sous forme de lemmes et de théorèmes.

A. Problème NP-difficile

Dans ce qui suit, nous considérons le problème $F2/Coup_Opr(1), a_i, b_i = L_i = p, c_i/C_{max}$, que nous notons par $F2(a_i, b_i = L_i = p, c_i)$. Nous montrons que ce problème est NP-difficile en utilisant le problème de réduction avec cardinalités identiques connu NP-difficile [10].

Le problème de partition avec cardinalités identiques (PCI) est défini comme suivant : étant donné un ensemble $E = \{e_1, e_2, \ldots, e_{2n}\}$ de $2n$ entiers positifs tel que $\sum_{i=1}^{2n} e_i = 2B$ et B un nombre entier. Existe t-il une partition de E en deux sous ensembles disjoints E_1 et E_2 tels que $\sum_{i \in E_1} e_i = \sum_{i \in E_2} e_i = B$ et $|E_1| = |E_2| = n$? Ce problème reste NP-difficile pour chaque $e_i > 1$. Dans notre preuve, nous supposons que tous les $e_i > 1$.

Etant donné une instance quelconque du problème PCI, nous construisons une instance τ du problème $F2(a_i, b_i = L_i = p, c_i)$ avec l'ensemble de $(4n+2)$ tâches comme suit :

- $2n$ tâches de types U, notées par U_i ;
- n tâches identiques, notées par V ;
- $n+1$ tâches identiques, notées par W ;
- Une seule tâche notée T ;

Pour toutes les tâches, on pose $L_i = b_i = p$, $i = 1, \ldots, 4n+2$ avec $p > B$. Les valeurs des durées opératoires des tâches sont données dans la table suivante :

Tâches	a_i	c_i
$U_i, i = 1, \ldots, 2n$	$p - e_i$	e_i
V	$p+1$	$4p$
W	p	0
T	$B + p + 1 - n$	$(4n+1)p - 2B$

Table I
Durées opératoires de tâches

Nous montrons que le problème PCI a une solution si seulement si le problème d'ordonnancement admet une solution S avec un $C_{max}(S) \leq y$, où $y = 4(2n+1)p + 1$. Alors, existe t-il un ordonnancement réalisable pour le problème $F2(a_i, b_i = L_i = p, c_i)$ avec un makespan inférieur ou égale à y ? Pour la preuve, nous avons établi les résultats suivants.

Lemme 1: Si le problème PCI a une solution alors il existe un ordonnancement S avec $C_{max}(S) \leq y$.

Preuve. Le problème de partition avec cardinalités identiques a une solution, i.e. qu'il existe une partition E_1 et E_2 de E, tel que $\sum_{i \in E_1} e_i = \sum_{i \in E_2} e_i = B$ où les ensembles E_1 et E_2 contient chaqu'un exactement n éléments. Alors nous pouvons ordonnancer les tâches correspondantes de l'instance τ comme suit.
Soient J_1 et J_2 les deux sous ensembles de $U-$tâches correspondants aux ensembles E_1 et E_2, respectivement. Alors, l'ordonnancement S existe lorsque la date de fin de traitement $C_{max}(S)$ de l'ordonnancement S est égale à $4(2n+1)p+1$. Dans cet ordonnancement, les $U-$tâches de $J_1(J_2)$ sont entrelacées avec les $V-$tâches ($W-$tâches), et une tâche W est entrelacée avec la tâche T. Soient $(VU)-$tâches $((VU)_1, \ldots, (VU)_n)$, $(WU)-$tâches $((WU)_1, \ldots, (WU)_n)$ et $(WT)-$tâche, les tâches composées obtenues en entrelaçant les opérations. L'ordonnancement S est construit comme suit (voir figure 1) : commençant à ordonnancer $(VU)-$tâches, suivies de $(TW)-$tâche et nous terminons par $(WU)-$ tâches. L'ordre des tâches composées des ensembles $(VU)-$tâches, et $(WU)-$tâches dans l'ordonnancement S peut être choisi dans n'importe quel ordre. Dans cet ordonnancement, il n'existe pas de temps mort entre les tâches composées sur les deux machines M_1 et M_2, alors,

$$C_{max} = p+1+2p+4np+\sum_{i \in J_1} e_i+(4n+1)p-2B+\sum_{i \in J_2} e_i$$

$$C_{max} = 4p + 8np + 1$$

$$C_{max} = 4(2n+1)p + 1. \qquad \blacksquare$$

Lemme 2: Dans un ordonnancement réalisable S, toutes les tâches sont entrelacées.

Preuve. Supposons que dans un ordonnancement S, il existe au moins deux tâches non entrelacées. Considérons que les $U-$tâches sont indexées dans l'ordre décroissant des e_j de ses durées de traitement a_j. Du moment que la durée totale de traitement des tâches composées sur la première machine est une borne inférieure du makespan, alors le meilleur entrelacement des opérations qui minimise la durée totale de traitement sur la première machine est une borne inférieure de l'ordonnancement S. Considérons l'entrelacement des opérations suivant : n tâches composées (VW), une tâche composée (TW), et $(n-1)$ tâches composées (U_iU_k) avec $i = n, \ldots, 2n-2$; $k = 1, \ldots, n-1$, et les tâches U_{2n-1} et U_{2n} sont ordonnancées seules, respectivement. Il est clair que cet entrelacement d'opérations de tâches sur la première machine fournit une durée de traitement totale minimale

lorsque exactement deux tâches ne sont pas entrelacées, ainsi :

$$C_{max}(S) \geq (B+p+1-n+3p)+(4np+n)+n*4p-\sum_{i=n-1}^{2n-2} e_i$$
$$-e_{2n-1}-e_{2n}$$
$$\geq 4(2n+1)p+1+2p+B-\sum_{i=n-1}^{2n} e_i$$
$$\geq y+2p+B-\sum_{i=n-1}^{2n} e_i$$

Tant que $p > B$ et $\sum_{i=n-1}^{2n} e_i < 2B$, alors $2p+B-\sum_{i=n-1}^{2n} e_i > 0$. Donc, $C_{max}(S) > y$. Par conséquent, dans une solution réalisable, toutes les tâches sont entrelacées. ■

Lemme 3: Dans un ordonnancement réalisable S, il n'existe pas de tâches composées $(UU)-$tâches.

Preuve. Soit S un ordonnancement contenant une tâche composée de $(UU)-$tâche. Supposons que cette $(UU)-$tâche est composée d'une pair (U_i, U_j) dans laquelle U_i est entrelacée avec U_j et les autres $U-$tâches sont entrelacées avec les autres types : $V-$tâches, $W-$tâches et $T-$tâche. Nous distinguons les cas suivants :

a. La tâche T est entrelacée avec une tâche de type $W-$tâches, alors il reste (n) $W-$tâches, (n) $V-$tâches et $(2n-2)$ $U-$tâches. Suivant ces tâches restantes, deux façons sont possibles pour les entrelacer, à savoir :
   1) $(TW), (WW), (VU_k)_{k\in I_1}, (U_kW)_{k\in I_2}$, où $|I_1| = n, |I_2| = n-2$ et $I_1 \cup I_2 \cup \{i,j\} = \{1,\ldots,2n\}$.
   2) $(TW), (VW), (VU_k)_{k\in I_1}, (U_kW)_{k\in I_2}$, où $|I_1| = n-1, |I_2| = n-1$ et $I_1 \cup I_2 \cup \{i,j\} = \{1,\ldots,2n\}$.

b. La tâche T est entrelacée avec une tâche de type $U-$tâche, alors il reste n $V-$tâches, $(n+1)W-$tâches et $(2n-3)$ $U-$tâches. De nouveau, suivant ces tâches restantes, trois façons sont possibles pour les entrelacer, à savoir :
   1) $(TU_r), (WW), (WW), (VU_k)_{k\in I_1}, (U_kW)_{k\in I_2}$, où $|I_1| = n, |I_2| = n-3$ et $I_1 \cup I_2 \cup \{i,j,r\} = \{1,\ldots,2n\}$.
   2) $(TU_r), (WW), (VW), (VU_k)_{k\in I_1}, (U_kW)_{k\in I_2}$, où $|I_1| = n-1, |I_2| = n-2$ et $I_1 \cup I_2 \cup \{i,j,r\} = \{1,\ldots,2n\}$.
   3) $(TU_r), (VW), (VW), (VU_k)_{k\in I_1}, (U_kW)_{k\in I_2}$, où $|I_1| = n-2, |I_2| = n-1$ et $I_1 \cup I_2 \cup \{i,j,r\} = \{1,\ldots,2n\}$.

Après le calcul de la valeur du makespan et de la borne inférieure de chaque cas, nous obtenons $LB > y$. Ainsi S n'est pas une solution réalisable avec $C_{max}(S) \leq y$. ■

Lemme 4: Dans un ordonnancement réalisable S, la tâche T est entrelacée avec la tâche de type W.

Preuve. Supposons qu'il existe un ordonnancement S dans lequel T est entrelacée avec $U-$tâche. Soit U_r la tâche $U-$tâche entrelacée avec $T-$tâche. A partir les lemmes 2 et 3, les seules tâches composées sont :

1) $(VU_k)_{k\in I_1}, (U_kW)_{k\in I_2}$ où $|I_1| = n-2$, $|I_2| = n+1$ et $I_1 \cup I_2 \cup \{r\} = \{1,\ldots,2n\}$.
2) $(VU_k)_{k\in I_1}, (U_kW)_{k\in I_2}$, (VW) où $|I_1| = n-1$, $|I_2| = n$ et $I_1 \cup I_2 \cup \{r\} = \{1,\ldots,2n\}$.
3) $(VU_k)_{k\in I_1}, (U_kW)_{k\in I_2}$, (WW) où $|I_1| = n$, $|I_2| = n-1$ et $I_1 \cup I_2 \cup \{r\} = \{1,\ldots,2n\}$.

Pour les cas précédents, les séquences optimales sont :

- Case 1. $S = \langle (VU_k)_{k\in I_1}, (TU_r), (U_kW)_{k\in I_2} \rangle$
- Case 2. $S = \langle (VU_k)_{k\in I_1}, (TU_r), (VW), (U_kW)_{k\in I_2} \rangle$
- Case 3. $S = \langle (VU_k)_{k\in I_1}, (TU_r), (U_kW)_{k\in I_2}, (WW) \rangle$

Similairement à la preuve du lemme 3, il est facile de montrer que pour chaque cas cité ci-dessus, $C_{max}(S) > y$. Alors dans un ordonnancement S, la tâche T est entrelacée avec la tâche W. ■

Lemme 5: Si le problème d'ordonnancement admet une solution avec $C_{max}(S) \leq y$, alors le problème PCI a une solution.

Preuve. A partir des lemmes $2, 3$ et 4, l'unique couple d'opérations de tâches entrelacées dans l'ordonnancement S est $(VU_k)_{k\in I_1}$, (TW) et $(U_kW)_{k\in I_2}$ tel que $|I_1| = n, |I_2| = n$ et $I_1 \cup I_2 = \{1,\ldots,2n\}$. La séquence optimale de ces tâches composées est : $S' = \langle (VU_k)_{k\in I_1}, (TW), (U_kW)_{k\in I_2} \rangle$. Les valeurs de X_1, X_2, Y_1 et Y_2 de cette séquence sont :

$$X_1 = 4np+p+1+B, \qquad X_2 = 4np+p-\sum_{k\in I_2} e_k.$$
$$Y_1 = 4np+\sum_{k\in I_1} e_k, \qquad Y_2 = 4np+p-2B+\sum_{k\in I_2} e_k.$$

La borne inférieure de $C_{max}(S)$ est :

$$LB = X_1 + \max\{X_2, Y_2\}$$

$$LB = y + \max\{B - \sum_{k\in I_2} e_k, \sum_{k\in I_2} e_k - B\}.$$

Tant que $C_{max}(S) \leq y$, alors $\max\{B - \sum_{k\in I_2} e_k, \sum_{k\in I_2} e_k - B\} = 0$. Ainsi, $\sum_{k\in I_2} e_k = B$ et $\sum_{k\in I_1} e_k = B$. Donc nous obtenons la solution pour le problème PCI. ■

A partir des lemmes 1 et 5, nous avons le résultat suivant :

Théorème 1: Le problème $F2/Coup-Oper(1), a_i, b_i = L_i = p, c_i/C_{max}$ est NP-difficile.

B. Le problème $F2/Coup-Oper(1), a_i = a, b_i = L_i = p, c_i/C_{max}$

Dans cette section, nous considérons le problème $F2(a_i = a, b_i = L_i = p, c_i)$. Deux cas sont à distinguer, à savoir si $a > p$, alors chaque tâche est ordonnancée séparément. Dans le cas où $a \leq p$, les tâches peuvent être entrelacées. Les lemmes

suivants sont utilisés pour proposer un algorithme polynomial qui permet de résoudre le problème $F2(a_i = a, b_i = L_i = p, c_i)$.

Lemme 6: Dans un ordonnancement optimal, toutes les tâches sont entrelacées sauf une tâche si n est un nombre impair.

Preuve. Soit S un ordonnancement optimal. Supposons qu'il existe dans S deux tâches J_i et J_j qui sont ordonnancés séparément, et supposons que la tâche J_i est ordonnancée avant J_j. Soit S' le nouveau ordonnancement construit à partir de S dans lequel J_j est entrelacée avec J_i pour former une nouvelle composante de tâche J_iJ_j, et elle est ordonnancée à la position J_i dans S. La durée de traitement totale qui est de S' sur M_1 est réduite de $p + a$ en respectant S et le makespan de S' ne décroit pas. Cependant, S' est un ordonnancement optimal. En répétant cette opération, nous obtenons un ordonnancement dans lequel les tâches sont entrelacées sauf la dernière tâche si le nombre de tâches est impair. ■

Lemme 7: Sur la deuxième machine, les tâches sont ordonnancées dans l'ordre décroissant des c_i.

Preuve. Soit S un ordonnancement optimal dans lequel le lemme 7 n'est pas vérifié. Soient J_i et J_j les deux premières tâches de S pour lesquelles $c_i \leq c_j$ et J_i est ordonnancée avant J_j. Considérons le nouveau ordonnancement S' dans lequel J_i et J_j sont permutées. Toutes les tâches composées ont la même durée de traitement sur la première machine, alors la durée totale de traitement de S' sur M_1 ne change pas, de plus, la durée de traitement des tâches composées contenants J_i dans S' n'est pas supérieure à celle des tâches de S. Ainsi, le nouveau ordonnancement S' est aussi optimal. En répétant cette opération, nous obtenons un ordonnancement avec les propriétés voulues. ■

Les lemmes précédents nous permettent de proposer l'algorithme 1 qui résout le problème $F2/Coup_Opr(1), a_i = a, b_i = L_i = p, c_i/C_{max}$.

Algorithm 1: algorithme 1

début
- **si** $a > p$ **alors**
  - Appliquer l'algorithme de Johnson telles que les durées de traitement des tâches sur la première et la deuxième machine sont : $p_{1i} = a + 2p$ et $p_{2i} = c_i$, $i = 1, \ldots, n$, respectivement.
- **sinon**
  - Indexer les tâches dans l'ordre décroissant de leur c_i, $i = 1, \ldots, n$.
  - Construire les tâches composées $T_i = (J_{2i-1}J_{2i})$, $i = 1, \ldots, n/2$, si n est pair, ou $T_i = (J_{2i-1}J_{2i})$, $i = 1, \ldots, (n-1)/2$ et $T_{\frac{n-1}{2}+1} = (J_n)$ si n est impair.
  - Ordonnancer les tâches composées T_i suivant l'ordre de leurs indices, le plutôt possible, sur les deux machines.
- **fin**

fin

Théorème 2: L'algorithme 1 fournit une solution optimale pour le problème $F2(a_i = a, b_i = L_i = p, c_i)$ en $O(nlogn)$.

Preuve. Si $a > p$ les tâches ne peuvent pas être entrelacées, alors l'ordonnancement optimal est obtenu en appliquant l'algorithme de Johnson [10] sur la liste des tâches. Si $a \leq p$, alors suivant les lemmes 6 et 7, toutes les tâches sont entrelacées sauf la dernière tâche si n est impair, et les tâches sont ordonnancées dans l'ordre décroissant de leurs durées de traitement sur la deuxième machine. L'algorithme 1 fournit un ordonnancement en respectant les résultats des lemmes 6 et 7 et cette solution est donnée en $O(nlogn)$. ■

C. Le problème $F2/Coup - Oper(1), a_i, b_i = L_i = p, c_i = c/C_{max}$

Nous considérons le problème $F2(a_i, b_i = L_i = p, c_i = c)$. Soient $K_1 = \{J_i/a_i > p\}$ et $K_2 = \{J_i/a_i \leq p\}$, et soient n_1 et n_2 les tailles des ensembles K_1 et K_2, respectivement. Nous supposons que les tâches de K_1 et K_2 sont indexées dans l'ordre décroissant de leurs a_i. Les tâches de K_1 ne peuvent être entrelacées qu'avec les tâches de K_2 où les tâches de K_1 sont en première position dans les tâches composées. Cependant, une tâche de K_2 peut être entrelacée avec une tâche de K_1 ou avec une autre tâche de K_2. Soit S_l l'ensemble des tâches dans lequel l tâches de K_1 sont entrelacées avec l tâches de K_2, $0 \leq l \leq \min\{n_1, n_2\}$, et le reste des tâches de K_1 restent non entrelacées.

Les résultats suivants sont utilisés pour définir un algorithme polynomial.

Lemme 8: Il existe un ordonnancement optimal de l'ensemble S_l tel que les l dernières tâches de K_2 sont entrelacées avec les l premières tâches de K_1.

Preuve. Pour la preuve, deux cas sont à distinguer à savoir :

1) les l dernières tâches de K_2 sont entrelacées avec les tâches de K_1.
2) les tâches de K_2 sont entrelacées avec les l premières tâches de K_1.

■

Dans ce qui suit, nous supposons que l'ensemble S_l satisfait le lemme 8.

Lemme 9: Il existe un ordonnancement optimal de l'ensemble S_l tel que :

1) le reste de toutes les tâches de K_2 sont entrelacées entre elles, sauf une tâche si leurs nombre est impair ;
2) les tâches $J_1^{K_2}, \ldots, J_{\lceil \frac{n_2-l}{2} \rceil}^{K_2}$ sont aux premières positions dans les nouvelles tâches composées.

A partir des lemme 8 et 9, nous avons le résultat suivant :

Lemme 10: L'algorithme de Mitten fournit un ordonnancement optimal de l'ensemble S_l en $O(n)$.

Considérons l'algorithme suivant :

Algorithm 2: algorithme 2

début
- Ranger les tâches de K_1 et K_2 dans l'ordre croissant de leurs a_i. Soit S^* un ordonnancement optimal. Poser $S^* = \emptyset$ et $C_{max}(S^*) = +\infty$.
- **pour** $l := 0$ *à* $\min\{n_1, n_2\}$ **faire**
  - Entrelacer les l dernières tâches de K_2 avec les l premières tâches de k_1.
  - Entrelacer le reste des tâches de K_2 entre elles, telles que les tâches $J_1^{K_2}, \ldots, J^{K_2}_{\lceil \frac{n_2-l}{2} \rceil}$ sont à la première position de la nouvelle tâche composée.
  - Appliquer l'algorithme de Mitten pour ordonnancer les nouvelles tâches. Soit S_l^* l'ordonnancement obtenu.
  - **si** $C_{max}(S_l^*) < C_{max}(S^*)$ **alors**
    - $S^* := S_l^*$ et $C_{max}(S^*) := C_{max}(S_l^*)$
  - **fin**
- **fin**

fin

Théorème 3: L'algorithme 2 résout le problème $F2(a_i, b_i = L_i = c_i = p)$ en $O(n^2 logn)$.

Preuve. Dans l'algorithme 2, l'ensemble des tâches composées construit satisfait les lemmes 8 et 9 pour chaque valeur de l. De plus, l'algorithme 2 selectionne la meilleure solution parmi les valeurs de l. Alors, la solution optimale est obtenue pour le problème $F2/Coup - Opr(1), a_i, b_i = L_i = p, c_i = c/C_{max}$. Clairement, l'algorithme 2 s'exécute en $O(n^2 logn)$. ■

D. Le problème $F2/Coup - Oper(1), a_i = L_i = p, b_i, c_i = c/C_{max}$

Dans cette section, nous considérons le problème $F2(a_i = L_i = p, b_i, c_i = c)$. Similairement au problème précédent, nous décomposons l'ensemble de tâches en deux sous ensembles $K_1 = \{J_i/b_i > p\}$ et $K_2 = \{J_i/b_i \leq p\}$. Soient n_1 et n_2 les tailles des ensembles K_1 et K_2, respectivement. Nous supposons que les tâches de K_1 et K_2 sont indexées dans l'ordre croissant de leurs b_i. Les tâches de K_1 peuvent être entrelacées uniquement avec les tâches de K_2 où une tâche de K_1 est dans la deuxième position de la tâche composée. Cependant, une tâche de K_2 peut être entrelacée avec la tâche de K_1 oubien avec une autre tâche de K_2.
Soit S_l l'ensemble de toutes les tâches dans lequel l tâches de K_1 sont entrelacées avec l tâches de K_2, $0 \leq l \leq \min\{n_1, n_2\}$, et le reste des tâches de K_1 restent seules. Ainsi, nous obtenons les résultats suivants :

Lemme 11: Il existe un ordonnancement optimal de l'ensemble S_l telles que les l dernières tâches de K_2 sont entrelacées avec les l premières tâches de K_1.

Preuve. La preuve est similaire à la preuve du lemme 8. ■

Lemme 12: Il existe un ordonnancement optimal de l'ensemble S_l tel que :

1) le reste de toutes les tâches de K_2 sont entrelacées entre elles, sauf une tâche si leurs nombre est impair ;
2) les tâches $J_1^{K_2}, \ldots, J^{K_2}_{\lceil \frac{n_2-l}{2} \rceil}$ sont aux premières positions dans les nouvelles tâches composées.

Preuve. La preuve est similaire à la preuve du lemme 9. ■

Pour la résolution de ce dernier problème, nous proposons un algorithme (algorithme 3) identique à l'algorithme 2 qui résout le problème $F2(a_i, L_i = b_i = p, c_i = c)$, sauf que dans cet algorithme, les tâches de K_1 et K_2 sont rangées dans l'ordre croissant de leurs b_i. Alors, nous avons le résultat suivant.

Théorème 4: L'algorithme 3 résout le problème $F2(a_i = L_i = p, b_i, c_i = c)$ en $O(n^2 logn)$.

Preuve. La preuve est identique à celle du problème $F2(a_i, b_i = L_i = p, c_i = c)$ dans le théorème 3. ■

IV. Heuristiques

Pour la résolution du problème $F2/Coup - Oper(1), a_i, L_i, b_i, c_i/C_{max}$, prouvé NP-difficile dans ref, nous avons proposé plusieurs heuristiques. Dans ce travail, nous présentons celles que nous estimons bonnes. L'heuristique J_P se base sur la règle de johnson[10] pour déterminer la séquence de tâches à ordonnancer. Ce qui est du calcul du makespan, nous introduisons la méthode en parallèle en considérant les tâches qui s'entrelacent. Les heuristiques $Lpt_a_i_P$ et $Lpt_b_i_P$ se basent sur la règle LPT des a_i et b_i respectivement pour déterminer la séquence de tâches à ordonnancer. L'heuristique $Spt_L_i_P$ se base sur la régle SPT des L_i. Pour le calcul du makespan de ces trois heuristiques, nous introduisons la méthode en parallèle en considérant les tâches à entrelacer. La dernière heuristique H^* se base sur l'ordonnancement de tâches suivant la règle LPT des a_i et b_i.

Heuristique 1 (Lpta_iP,Lptb_iP)
1) Ranger les tâches suivant la règle LPT (Longuest Processing Time) des a_i (respectivement l_i, b_i, c_i).
2) Utiliser la méthode en série (respectivement en parallèle) pour déterminer la séquence de tâches à ordonnancer en les entrelaçant.

Heuristique 2 (Spta_iP)
1) Ranger les tâches suivant la règle SPT (Shortest Processing Time) des a_i (respectivement l_i, b_i, c_i).
2) Utiliser la méthode en série (respectivement en parallèle) pour déterminer la séquence de tâches à ordonnancer en les entrelaçant.

Heuristique 3 (JP)
1) Pour chaque tâche, poser : $p_{1i} = a_i + l_i + b_i$ et $p_{2i} = c_i$.
2) Ranger les tâches suivant la règle de Johnson.
3) Utiliser la méthode en série (respectivement en parallèle) pour déterminer la séquence de tâches à ordonnancer en les entrelaçant.

Heuristique 4 (H^*)
1) Former deux listes de tâches L_1 et L_2 en ordonnançant les tâches suivant la régle LPT (Longuest Processing Time) des a_i et b_i, respectivement.

2) Pour chaque tâche de L_1, parcourir la liste des tâches L_2 et entrelacer la tâche de L_1 avec la première tâche non entrelacée de L_2.

V. Expérimentations numériques

Pour évaluer la performance des heuristiques proposées ci-dessus, nous avons réalisé des expérimentations numériques. Alors, nous avons testé les heuristiques pour les différentes valeurs de tâches de l'ensemble $\{10, 30, 50, 100, 250, 500, 1000\}$. Les durées de traitement et les valeurs du délai pour les différents cas sont générés aléatoirement suivant la loi uniforme. Ces durées prennent leurs valeurs dans les intervalles suivants : $a_i, L_i, b_i, c_i \in [1, 50]$; $a_i, L_i, b_i, c_i \in [1, 100]$; $a_i, L_i, b_i, c_i \in [50, 100]$; et $a_i, L_i, b_i, \in [100, 150], c_i \in [1, 50]$. Les tables II, III, IV et V représentent les résultats obtenus à partir des expérimentations. Pour chaque valeur de tâches, nous avons généré 100 instances sur lesquelles nous avons appliqué les heuristiques. Pour chaque heuristique, nous avons noté le nombre de fois où la solution trouvée est meilleure (C_{max}) et le nombre de fois où l'optimum est atteint (opt). Nous avons également calculé la durée moyenne d'exécution $\overline{time}$, la déviation moyenne $\overline{dev}$ et la déviation maximale $(mdev)$ de chaque heuristique. La déviation d'une heuristique est calculé par la formule $dev(H) = \frac{C_{max}(H) - LB}{LB}$.

D'après les expérimentations réalisées, nous avons remarqué que l'heuristique $LPTa_iP$ donne de meilleurs valeurs de C_{max} pour $a_i, L_i, b_i, c_i \in [1, 50]$; et pour $a_i, L_i, b_i, c_i \in [1, 100]$ par rapport aux autres heuristiques. Pour les valeurs de $a_i, L_i, b_i, c_i \in [50, 100]$; et $a_i, L_i, b_i, \in [100, 150], c_i \in [1, 50]$, initialement, ç-à-d pour un petit nombre de tâches, l'heuristique $SPTL_iP$ fournie de meilleurs résultats de C_{max} en la comparant avec les autres heuristiques, mais avec la croissance du nombre de tâches, c'est l'heuristique $LPTa_iP$ qui fournit de meilleurs résultats de C_{max}. Les durées moyennes d'exécution des heuristiques croient avec la croissance du nombre de tâches. Nous avons également constaté que les heuristiques qui utilisent les algorithmes parallèles pour le calcul du makespan nécessitent une durée d'exécution moins importante par rapport à celles qui utilisent les algorithmes sérielles.

VI. Conclusion

Dans ce travail, nous avons présenté le problème d'ordonnancement de type flowshop avec des tâches couplées. Notre problème consiste en un problème du flowshop à deux machines avec des opérations couplées sur la première machine séparées par un délai exact, et d'une seule opération sur la deuxième machine. Notre objectif est de minimiser le makespan. Cependant, nous avons montré qu'un problème est NP-difficile comme nous avons exposé des sous problèmes polynomiaux. Pour sa résolution, nous avons proposé des heuristiques avec des expérimentations numériques. D'après les tests numériques, nous avons constaté que l'heuristque

		$a_i, l_i, b_i, c_i \in [1, 50]$				
		JP	H^*	Lpta_iP	Lptb_iP	SptL_iP
10	C_{max}	10	11	7	6	24
	opt	0	0	0	0	0
	$time$	0.1	0.4	0.2	0.1	0
	dev	0.242	0.283	0.242	0.241	0.206
	mdev	0.468	0.745	0.408	0.491	0.435
30	C_{max}	4	17	21	16	35
	opt	0	0	0	0	0
	$time$	0.1	0.9	0.4	0.4	1
	dev	0.205	0.252	0.166	0.174	0.165
	mdev	0.367	0.528	0.303	0.336	0.378
50	C_{max}	0	9	46	16	26
	opt	0	0	0	0	0
	$time$	0.7	1.1	0.4	0.7	0.8
	dev	0.197	0.243	0.156	0.158	0.157
	mdev	0.293	0.498	0.195	0.218	0.239
100	C_{max}	0	3	63	21	13
	opt	0	0	0	0	0
	$time$	1.2	1.9	1	1.5	1.4
	dev	0.176	0.227	0.125	0.135	0.144
	mdev	0.226	0.428	0.181	0.172	0.232
250	C_{max}	0	1	93	5	1
	opt	0	0	0	0	0
	$time$	4.95	28.21	4.69	4.47	4.28
	dev	0.159	0.215	0.103	0.124	0.137
	mdev	0.205	0.340	0.138	0.159	0.191
500	C_{max}	0	0	100	0	0
	opt	0	0	0	0	0
	$time$	11.5	38.64	11.2	11.89	11.48
	dev	0.149	0.206	0.088	0.124	0.134
	mdev	0.192	0.283	0.106	0.145	0.187
1000	C_{max}	0	0	100	0	0
	opt	0	0	0	0	0
	$time$	38.02	234.4	40.34	40.32	42.21
	dev	0.144	0.201	0.079	0.129	0.134
	mdev	0.177	0.250	0.096	0.145	0.169

Table II
Résultats des tests

		$a_i, l_i, b_i, c_i \in [1, 100]$				
		JP	H^*	Lpta_iP	Lptb_iP	SptL_iP
10	C_{max}	13	11	3	1	18
	opt	0	3	4	6	11
	$time$	0.12	0.16	0.15	0.3	0.15
	dev	0.263	0.320	0.263	0.274	0.255
	mdev	0.450	0.798	0.410	0.406	0.484
30	C_{max}	6	13	28	17	25
	opt	0	0	0	0	0
	$time$	0.31	0.94	0.31	0.47	0.31
	dev	0.215	0.258	0.176	0.189	0.181
	mdev	0.432	0.624	0.336	0.369	0.414
50	C_{max}	1	17	34	17	29
	opt	1	0	0	0	0
	$time$	0.46	0.94	1.11	0.3	0.31
	dev	0.201	0.244	0.152	0.159	0.156
	mdev	0.275	0.508	0.232	0.234	0.263
100	C_{max}	0	5	54	20	21
	opt	0	0	0	0	0
	$time$	1.11	2.33	1.39	0.31	1.51
	dev	0.179	0.228	0.127	0.136	0.145
	mdev	0.254	0.391	0.166	0.183	0.219
250	C_{max}	0	0	86	12	2
	opt	0	0	0	0	0
	$time$	2.82	3.56	2.33	2.62	2.98
	dev	0.165	0.223	0.111	0.123	0.138
	mdev	0.208	0.328	0.140	0.139	0.188
500	C_{max}	0	0	100	0	0
	opt	0	0	0	0	0
	$time$	8.6	33.1	8.1	8.1	8.4
	dev	0.153	0.212	0.095	0.115	0.137
	mdev	0.186	0.287	0.112	0.133	0.173
1000	C_{max}	0	0	100	0	0
	opt	0	0	0	0	0
	$time$	32.61	106.8	35.47	35.34	32.45
	dev	0.148	0.207	0.085	0.119	0.136
	mdev	0.167	0.266	0.098	0.132	0.161

Table III
Résultats des tests

		$a_i, l_i, b_i, c_i \in [50, 100]$				
		JP	H^*	Lpt$a_i P$	Lpt$b_i P$	Spt$L_i P$
10	C_{max}	9	3	12	2	13
	opt	0	0	0	0	0
	$time$	0.1	0.4	0.1	0.1	0.2
	dev	0.180	0.219	0.173	0.160	0.131
	mdev	0.350	0.387	0.331	0.332	0.359
30	C_{max}	8	5	14	15	52
	opt	0	0	0	0	0
	$time$	0.32	0	0.31	0.15	0.46
	dev	0.135	0.197	0.109	0.112	0.078
	mdev	0.229	0.334	0.209	0.229	0.189
50	C_{max}	0	0	16	9	74
	opt	0	0	0	0	0
	$time$	0.8	1	0.7	0.7	0.8
	dev	0.117	0.185	0.089	0.098	0.063
	mdev	0.184	0.289	0.164	0.168	0.123
100	C_{max}	2	0	18	7	73
	opt	0	0	0	0	0
	$time$	1.53	2.91	1.62	1.44	1.6
	dev	0.099	0.171	0.073	0.079	0.055
	mdev	0.154	0.246	0.122	0.136	0.076
250	C_{max}	0	0	28	1	71
	opt	0	0	0	0	0
	$time$	4	27.62	4.28	4.06	4.05
	dev	0.077	0.153	0.054	0.069	0.047
	mdev	0.102	0.193	0.075	0.093	0.064
500	C_{max}	0	0	59	0	41
	opt	0	0	0	0	0
	$time$	9.61	12.55	9.03	8.96	7.87
	dev	0.064	0.152	0.045	0.067	0.046
	mdev	0.077	0.184	0.062	0.086	0.057
1000	C_{max}	0	0	93	0	7
	opt	0	0	0	0	0
	$time$	24.23	156.8	25.52	27.75	26.27
	dev	0.056	0.144	0.037	0.070	0.046
	mdev	0.066	0.169	0.046	0.087	0.054

TABLE IV
RÉSULTATS DES TESTS

		$a_i, L_i, b_i \in [100, 150], c_i \in [1, 50]$				
		JP	H^*	Lpt$a_i P$	Lpt$b_i P$	Spt$L_i P$
10	C_{max}	3	11	13	5	21
	opt	0	3	4	6	11
	$time$	0.32	0.45	0.15	0.2	0.1
	dev	0.173	0.205	0.158	0.151	0.128
	mdev	0.365	0.495	0.739	0.376	0.368
30	C_{max}	2	9	19	13	46
	opt	0	0	0	0	0
	$time$	0.32	1.26	0.77	0.16	0.16
	dev	0.125	0.182	0.095	0.102	0.072
	mdev	0.229	0.331	0.253	0.253	0.231
50	C_{max}	1	0	13	15	72
	opt	0	0	0	0	0
	$time$	0.31	1.41	0.77	0.32	0.31
	dev	0.105	0.168	0.082	0.085	0.052
	mdev	0.166	0.286	0.151	0.151	0.126
100	C_{max}	1	0	12	4	83
	opt	0	0	0	0	0
	$time$	1.99	21.07	1.26	0.93	1.39
	dev	0.079	0.152	0.061	0.068	0.038
	mdev	0.149	0.232	0.110	0.121	0.072
250	C_{max}	0	0	14	1	86
	opt	0	0	0	0	0
	$time$	2.34	65.02	3.59	5.03	4.07
	dev	0.059	0.141	0.045	0.058	0.029
	mdev	0.083	0.195	0.068	0.097	0.041
500	C_{max}	0	0	27	0	73
	opt	0	0	0	0	0
	$time$	13.89	102.8	14.36	14.16	13.8
	dev	0.045	0.134	0.035	0.057	0.027
	mdev	0.059	0.160	0.049	0.078	0.035
1000	C_{max}	0	0	57	0	43
	opt	0	0	0	0	0
	$time$	26.59	76.23	27.35	28.01	26.7
	dev	0.038	0.130	0.027	0.057	0.027
	mdev	0.046	0.153	0.039	0.074	0.032

TABLE V
RÉSULTATS DES TESTS

à base de la règle LPT et qui utilise un algorithme parallèle, $LPTa_iP$, pour le calcul du makespan est plus performante par rapport aux autres heuristiques dans le cas où $a_i, L_i, b_i, c_i \in [1, 50]$ et $a_i, L_i, b_i, c_i \in [1, 100]$. Dans le cas où $a_i, L_i, b_i, c_i \in [50, 100]$; et $a_i, L_i, b_i, \in [100, 150], c_i \in [1, 50]$, l'heuristique $SPTL_iP$, qui utilise un algorithme parallèle, est plus performante par rapport aux autres heuristiques pour un petit nombre de tâches, et pour un grand nombre de tâches, c'est l'heuristique $LPTa_iP$ qui est la plus performante.

RÉFÉRENCES

[1] A.A. Ageev and A.E. Baburin, *Approximation algorithms for UET scheduling problems with exact delays*, Operations Research Letters 35 : 533-540, 2007.

[2] A.A. Ageev and A.V. Kononov, *Approximation Algorithms for Scheduling Problems with Exact Delays*, In WAOA : 1-14, 2006.

[3] D. Ahr, J. Békési, G. Galambos, M. Oswald and G. Reinelt, *An exact algorithm for scheduling identical coupled tasks*, Mathematical Methods of Operational Research 59 : 193-203, 11,June 2004.

[4] J. Blazewicz, K. Ecker, T. Kis, C.N. Potts, M. Tanas and J. Whitehead, *Scheduling of coupled tasks with unit processing times*, Journal of Scheduling 13 : 453-461, 2010.

[5] N. Brauner, G. Finke, V. Lehoux-Lebacque, C. Potts and J. Whitehead, *Scheduling of coupled tasks and one-machine no-wait robotic cells*, Computers and Operational Research 36(2) : 301-307, 2009.

[6] M. Dell'Amico, *Shop problems with two machines and time lags*, Operations Research 44 : 777-787, 1996.

[7] J. Fondrevelle, A. Oulamara and M.C. Portmann, *Permutation flow shop scheduling problems with time lags to minimize the weigted sum of machine completion times*, International Journal of Production Economics 112 : 168-176, 2008.

[8] M.R. Garey and D.S. Johnson, *Computers and Intractability : A Guide to the Theory of NP- Completeness*, Victor Klee, ed. A Series of Books in the Mathematical Sciences. San Francisco, Calif. : W. H. Freeman and Co., 1979.

[9] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, *Optimization and approximation in deterministic sequencing and scheduling : a survey*, Annals of Discrete Mathematics 5 : 287-326, 1979.

[10] S.M. Johnson, *Optimal two and three stage production schedules with setup time included*, Naval Research Logistics Quarterly 1 : 61-67, 1954.

[11] J.K. Lenstra, Private communication, 1991.

[12] N. Meziani, A. Oulamara and M. Boudhar, *Problème du flowshop à deux machines avec des opérations couplées sur la première machine*, Operational Research Practice in Africa (ORPA'2015), USTHB, Algiers, Algeria : Avril 2015.

[13] L.G. Mitten, *Sequencing n Jobs on Two Jobs with Arbitrary Time Lags*, Management Science 5(3) : 293-298, 1958.

[14] A.J. Orman and C.N. Potts, *On the complexity of coupled-Task Scheduling*, Discrete Applied Mathematics 72 : 141-154, 1997.

[15] R.D. Shapiro, *Scheduling Coupled Tasks*, Naval Research logistics quartely 20 : 489-498, 1980.

[16] G. Simonin, R. Giroudeau and J.C. König, *Polynomial-time algorithms for scheduling problem for coupled-tasks in presence of treatment tasks*, Electronic Notes in Discrete Mathematics 36 : 647-654, 2010.

[17] W. Yu, H. Hoogeveen and J.K. Lenstra, *Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard*, Journal of Scheduling 7 : 333-348, 2004.

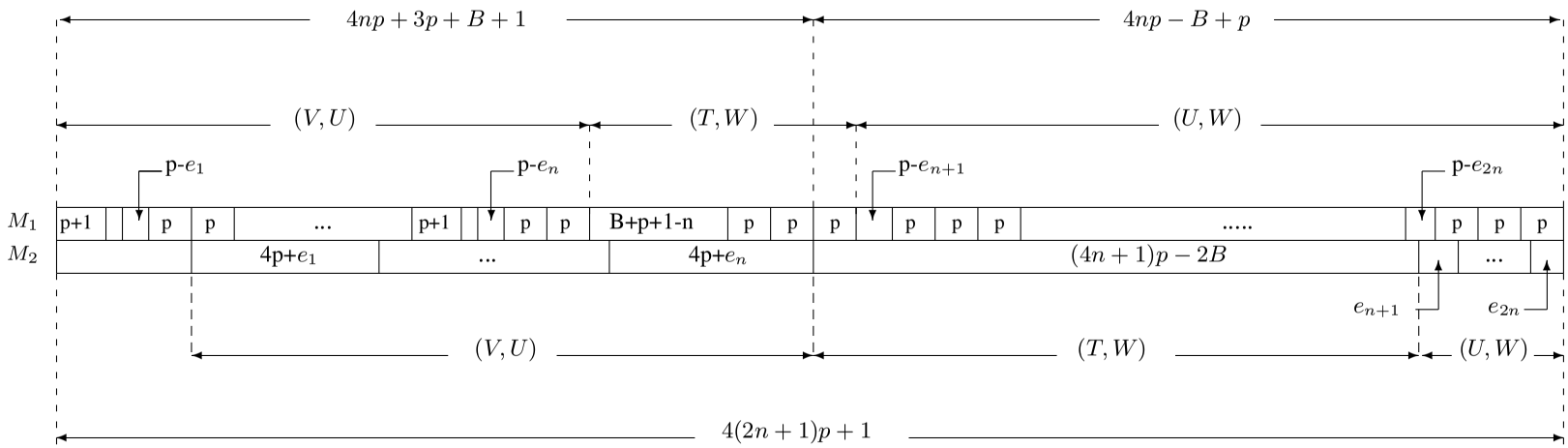

FIGURE 1. Ordonnancement S