Ordonnancement sur machines identiques en présence d'ouvriers spécialisés

Mourad Boudhar et Wafaa Labbi

Faculté de Mathématiques, Université USTHB,
BP 32 Bab-Ezzouar, El-Alia 16111, Alger, Algérie
mboudhar@usthb.dz et fawalab@yahoo.fr

Résumé On s'intéresse au problème d'ordonnancement de tâches non préemptibles et indépendantes sur machines parallèles identiques en présence d'ouvriers spécialisés. Chaque tâche doit subir, avant d'être exécuter sur une machine, un traitement particulier par un ouvrier spécialisé. Les machines identiques ainsi que les ouvriers spécialisés ne peuvent traiter qu'une seule tâche à la fois. Nous montrons que le problème général est NP-difficile et nous donnons quelques sous problèmes polynomiaux. Une méthode exacte basée sur la modélisation mathématique du problème et des heuristiques sont aussi présentées avec des résultats expérimentaux. Basées sur des instances générées aléatoirement, ces expérimentations nous permettent d'apprécier l'efficacité des méthodes proposées.

Mots clés Ordonnancement, machines identiques, serveurs, makespan.

1 Introduction

Dans la littérature, de nombreuses recherches traitent des problèmes d'ordonnancement à machines parallèles identiques. Fruits de ce vaste travail, plusieurs contraintes ont été considérées. Sans contraintes de préparation, le problème de minimisation de la date de fin de traitement C_{max} a été largement étudié, ce dernier est NP-difficile [7]. Plusieurs cas de ce problème ont été également étudiés avec différents types de contraintes, pour plus de détails le lecteur peut se référer à [2,3,4,5,6,10].

Nous traitons dans ce papier un problème d'ordonnancement à machines parallèles identiques. Il s'agit d'ordonnancer un ensemble de n tâches non préemptibles et indépendantes $T_1, T_2, ..., T_n$ sur un ensemble de m machines identiques $M_1, M_2, ..., M_m$ pour optimiser le makespan qui correspond à la date de fin de traitement de l'ensemble des tâches. Chaque tâche T_i (i=$\overline{1,n}$) nécessite un temps de traitement p_i et un temps de préparation S_i. Pour cette préparation, nous disposons de k ouvriers spécialisés, il est donc impossible de faire plus de k préparations en même temps. Ce problème sera noté : $Pm/S_i, k/C_{max}$.

Pour les problèmes à machines parallèles identiques en présence d'un seul serveur pour la préparation des tâches, Abdelkhodaee et Wirth [1] ont étudiés le problème à deux machines parallèles identiques dont l'objectif est de minimiser la date de fin de traitement des tâches. Ils ont montrés que le problème général est NP-difficile au sens fort et ils ont proposés une formulation mathématique en

nombres entiers, deux cas particuliers sont étudiés ainsi que deux heuristiques en $O(nlogn)$ sont présentés avec des expérimentations numériques. Koulamas [8] a traité le problème d'ordonnancement de deux machines parallèles Semi-automatiques pour minimiser le temps mort résultant de l'indisponibilité du robot avec la condition que ces deux machines partagent le même serveur (robot) pour la préparation des tâches, il a démontré que ce dernier est NP-difficile au sens fort, aussi il a développé une procédure de réduction pour le transformer en un problème plus petit. Kravchenko et Werner [9] ont traités le problème d'ordonnancement de m machines parallèles en présence d'un seul serveur dont l'objectif est de minimiser le makespan, ils ont présentés un algorithme pseudo-polynomial pour le cas de deux machines où les temps de préparation sont égaux à 1, ils ont aussi montrés que le problème général avec un nombre arbitraire de machines est NP-difficile. Zouba et al. [11] ont étudiés le problème d'ordonnancement non préemptif sur deux machines parallèles identiques en présence d'un seule opérateur pour minimiser le makespan. Le problème consiste à déterminer des intervalles de temps d'utilisation des différentes affectations de l'opérateur pour ordonnancer les tâches sur les deux machines.

$Pm/S_i, k/C_{max}$ peut trouver des applications dans les opérations d'assemblage de petites usines où un robot est monté entre deux lignes d'assemblage de sorte qu'il peut servir les deux lignes. Ce dernier peut trouver aussi des applications dans les systèmes de fabrication flexibles (FMSs) tel qu'un atelier flexible est un environnement de fabrication souple et automatisé, il est composé généralement de trois éléments principaux : un système de fabrication, un système de manutention (ou de transport) et un système d'information. Il peut être rencontré aussi dans le cas d'un atelier de fabrication cellulaire où on a une ou plusieurs cellules constituées de machines, et entre les cellules, un ou plusieurs robots sont chargés de transporter les tâches.

Cet article est organisé comme suit. Dans la section 2, nous proposons une modélisation mathématique et dans la section 3, nous donnons une analyse de la complexité du problème. Une borne inférieure ainsi qu'un sous problème polynomial sont identifiés dans la section 4. La section 5 est consacrée aux heuristiques de résolution. Des tests numériques sont réalisés à la section 6. Une conclusion prendra forme dans la dernière section.

2 Modélisation mathématique

Dans cette section, une modélisation mathématique est proposée. Pour ce faire, considérons les variables bivalentes X_{ijt} telles que :

$X_{ijt}=\begin{cases} 1 \text{ si la tâche } T_i \text{ débute sa préparation sur la machin } M_j \text{ à l'instant t} \\ 0 \text{ sinon.} \end{cases}$

pour $i = \overline{1,n}$; $j = \overline{1,m}$ et $t = \overline{0,H-1}$ avec H un temps limite qu'on peut estimer à la borne supérieure.

On veut minimiser la date de fin de traitement de l'ensemble des tâches notée μ. Le modèle mathématique ainsi développé s'écrit :

Min μ

$$\begin{cases} \sum_{j=1}^{m}\sum_{t=0}^{H-1} X_{ijt} = 1 & i = \overline{1,n} \quad (1) \\ \sum_{y=t}^{t+S_i+p_i-1} X_{ijy} \leq 1 & j = \overline{1,m};\ i = \overline{1,n};\ t = \overline{0,H-1} \quad (2) \\ \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{y=max\{0,t-S_i+1\}}^{t} X_{ijy} \leq k & t = \overline{0,H-1} \quad (3) \\ (\sum_{j=1}^{m}\sum_{t=0}^{H-1} tX_{ijt}) + S_i + p_i \leq \mu & i = \overline{1,n} \quad (4) \\ X_{ijt} \in \{0,1\} & j = \overline{1,m};\ i = \overline{1,n};\ t = \overline{0,H-1} \quad (5) \\ \mu \in IR & \quad (6) \end{cases}$$

La première contrainte assure que l'on assigne une tâche à une machine et une seule. La deuxième contraintes indique que dans l'intervalle du temps [t,t+S_i+p_i-1] on ne peut traiter qu'au plus une seule tâche. La troisième contraintes signifie qu'on ne peut pas faire plus de k préparation à un instant t donné. Et la dernière contrainte indique que la date de fin de traitement de chaque tâche doit être inférieure ou égale à μ. Le modèle mathématique associé requiert $nmH + 1$ variables et $2n + (mnH) + H$ contraintes. Par exemple : pour $n = 3$, $m = 2$ et $H = 5$, on a 31 variables et 41 contraintes.

Les tests réalisés en utilisant un solveur de programmation linéaire mixte sur des instances de petites tailles, confirme l'efficacité de notre modélisation.

3 Analyse de la complexité

Le problème à deux machines $P2/S_i, k/C_{max}$ est NP-difficile, car le problème $P2//C_{max}$ en est un cas particulier en prenant $S_i = 0$.

Théorème 1. *Le problème $P2/p_i = p, S_i, k = 2/C_{max}$ est NP-difficile.*

Démonstration. Considérons le problème de décision NP-complet suivant connu sous le nom de 2-partition : étant donnés n entiers positifs $a_1, a_2, ..., a_n$. Existe-t-il un sous-ensemble $A_1 \subseteq A$ ($A = \{a_1, a_2, ..., a_n\}$) tel que : $\sum_{a_i \in A_1} a_i = \sum_{a_i \in A \setminus A_1} a_i$?

Montrons que ce problème se réduit polynomialement au problème de décision suivant : Etant données n tâches T_1, T_2, ..., T_n indépendantes et non morcelables avec des temps de traitement $p_i = p = 0$, le temps nécessaire pour la préparation d'une tâche est $S_i = a_i$, $k = 2$ et $Y = \frac{1}{2}\sum_{i=1}^{n} a_i$. Existe-t-il un ordonnancement de ces n tâches sur deux machines de durée inférieure ou égale à Y ?

Ce problème appartient à la classe NP, car on peut vérifier en un temps polynomial qu'une affectation des tâches aux machines vérifie toutes les contraintes. Il est clair que la réduction précédente est polynomiale ($O(n)$). Nous prouvons que le problème 2-partition a une solution si et seulement si le problème d'ordonnancement a une solution.

Si le problème 2-partition a une solution, alors il existe un sous-ensemble $A_1 \subseteq A$ avec la propriété désirée : $\sum_{a_i \in A_1} a_i = \sum_{a_i \in A \setminus A_1} a_i$. Nous construisons une solution pour le problème d'ordonnancement comme suit : on prépare les tâches qui appartiennent à l'ensemble A_1 sur la machine M_1 et les tâches restantes, on les prépare sur la machine M_2. Donc, la durée de l'ordonnancement est égale à $Y = \frac{1}{2}\sum_{i=1}^{n} a_i$.

Si le problème d'ordonnancement a une solution de durée inférieure ou égale à Y, alors, chaque tâche n'est préparée que sur l'une des deux machines. Donc le problème 2-partition a une solution, en posant A_1 comme étant l'ensemble de tâches préparées sur la machine M_1 et $A \setminus A_1$ l'ensemble de tâches préparées sur la machine M_2. □

Théorème 2. *Le problème $P2/S_i = s, k = 2/C_{max}$ est NP-difficile.*

Démonstration. Soit le problème de décision 2-partition : étant donné un ensemble $A = \{a_1, a_2, ..., a_n\}$ d'entiers positifs. Existe-t-il un sous-ensemble $A_1 \subseteq A$ tel que : $\sum_{a_i \in A_1} a_i = \sum_{a_i \in A \setminus A_1} a_i$?

Montrons que ce problème se réduit polynomialement au problème à deux machines parallèles identiques, étant données n tâches $T_1, T_2, ..., T_n$ indépendantes et non morcelables avec des temps de préparation $S_i = s = 0$ et $k = 2$, chaque tâche T_i a un temps de traitement $p_i = a_i$, et un nombre $Y = \frac{1}{2}\sum_{i=1}^{n} a_i$. Existe-t-il un ordonnancement de ces n tâches sur deux machines de durée inférieure ou égale à Y ?

Ce problème appartient à la classe NP, car on peut vérifier en un temps polynomial qu'une affectation des tâches aux machines vérifie toutes les contraintes. Il est clair que la réduction précédente est polynomiale ($O(n)$). Nous prouvons que le problème 2-partition a une solution si et seulement si le problème d'ordonnancement a une solution.

Supposons que le problème 2-partition a une solution, donc il existe un sous-ensemble $A_1 \subseteq A$ tel que $\sum_{a_i \in A_1} a_i = \sum_{a_i \in A \setminus A_1} a_i$. Ainsi, on peut construire un ordonnancement pour le problème P2/S_i = s, k=2 /C_{max} en traitant les $|A_1|$ tâches sur la machine M_1 et les $|A \setminus A_1|$ tâches restantes sur la machine M_2. Donc, la durée de l'ordonnancement est égale à Y.

Supposons maintenant que le problème d'ordonnancement a une solution de durée inférieure ou égale à Y, alors chaque tâche n'est affectée qu'à l'une des deux machines de telle sorte que A_1 soit l'ensemble des tâches affectées à la machine M_1 et $A \setminus A_1$ l'ensemble des tâches affectées à la machine M_2. Donc, le problème 2-partition a une solution. □

Théorème 3. *Le problème $P2/S_i, k = 1/C_{max}$ est NP-difficile au sens fort.*

Démonstration. La preuve de ce théorème a été donnée dans [1]. Nous proposons une preuve plus facile : il suffit de constater que ce problème est équivalent au problème défini dans [8] par C. P. Koulamas où il cherche à minimiser le temps mort résultant de l'indisponibilité du robot. Or, minimiser le temps mort revient à minimiser le makespan. □

4 Borne inférieure et étude d'un sous problème polynomial

4.1 Bornes inférieure et supérieure

Nous proposons une borne inférieure pour le C_{max}.

Théorème 4. $\overline{M} = \max\{\lceil \frac{1}{m} \sum_{i=1}^{n}(p_i + S_i)\rceil, \max_{i=1}^{n}\{p_i + S_i\}\}$ *est une borne inférieure.*

Démonstration. Par l'absurde : Si $\overline{M} < \max_{i=1}^{n}\{p_i + S_i\}$, la tâche ayant le plus grand temps nécessaire pour son traitement $(p_i + S_i)$ sera traitée sur au moins deux machines simultanément, contradiction avec les hypothèses qu' à chaque instant, une tâche ne peut être exécutée que par une seule machine au plus, et à chaque instant, une machine ne peut exécuter qu'une seule tâche à la fois. D'où $\overline{M}$ est une borne inférieure pour le C_{max}.

Si $\overline{M} < \max\{\lceil \frac{1}{m} \sum_{i=1}^{n}(p_i + S_i)\rceil$, contradiction du fait que le C_{max} ne peut pas prendre une valeur inférieure à la somme des temps de traitement et les temps de préparation des tâches divisée par le nombre de machines. □

Si toutes les tâches sont traitées sur la même machine, on aura comme borne supérieure $\overline{S}$, tel que : $\overline{S} = \sum_{i=1}^{n}(p_i + S_i)$.

4.2 Problème $P2/p_i = p, S_i = s, k \leq 2/C_{max}$

Pour ce problème avec des temps de traitement et de préparation constants, nous proposons un algorithme polynomial pour le résoudre.

Algorithm A ;
début

- $\overline{M'}$= $\begin{cases} \frac{n(p+S)}{2} & \text{si } n \text{ est pair et } k = 2\,; \\ \frac{n(p+S)}{2} + S & \text{si } n \text{ est pair et } k = 1 \text{ avec } S \leq p\,; \\ n.S + p & \text{si } n \text{ est pair et } k = 1 \text{ avec } S > p\,; \\ \frac{(n+1)(p+S)}{2} & \text{si } n \text{ est impair et } k = 2\,; \\ \frac{(n+1)(p+S)}{2} & \text{si } n \text{ est impair et k} = 1 \text{ avec } S \leq p\,; \\ n.S + p & \text{si } n \text{ est impair et } k = 1 \text{ avec } S > p\,; \end{cases}$

- $t := 0$; $i := 1$; $j := 1$; $l := 1$;
- **répéter**
  si $(t + p_i + S_i) \leq \overline{M'}$ **alors**
  - Affecter la préparation de la tâche T_i à l'ouvrier l et la traiter sur la machine M_j à l'instant t ;
  - $t := t + p_i + S_i$; $i := i + 1$

  sinon **si** $k = 1$ **alors** $t := S$; $j := j + 1$
  sinon $t := 0$; $l := l + 1$; $j := j + 1$;
  fsi ;

  fsi ;

jusqu'à $i = n$;
fin.

Théorème 5. *L'algorithme A résout le problème* $P2/p_i = p, S_i = s, k \leq 2/C_{max}$ *en* $O(n)$.

Démonstration. La valeur $\overline{M'}$ constitue une borne inférieure pour la solution optimale. Comme les temps de traitement et les temps de préparation sont constants et égaux à p et s respectivement, on a :

Pour $k = 2$ et d'après la borne inférieure énoncée dans la proposition 3.1 précédente, nous avons $\overline{M'}$=max$\{\lceil \frac{1}{2}n(p+S)\rceil, p+S\}$=$\lceil \frac{1}{2}n(p+S)\rceil$=$\frac{n(p+S)}{2}$ si n est pair. Si n est impair, alors $C_{max} = \frac{(n-1)(p+S)}{2} + (p+S) = \frac{(n+1)(p+S)}{2}$ qui vient du fait que la préemption des tâches n'est pas autorisée. Il suffit donc de considérer le cas pair et d'ajouter le traitement de la n-ème tâche à la fin de l'ordonnancement.

Pour $k = 1$, nous avons un seul ouvrier. Donc la préparation simultanée de deux tâches n'est pas possible. Ainsi, la seconde machine sera libre pendant S unités de temps au début de l'ordonnancement.

Si le temps de préparation est supérieur au temps de traitement ($S \geq p$), alors la préparation des différentes tâches se fera en alterné sur les deux machines et chaque tâche sera traitée entre deux préparations successives. On obtient donc $n.S$ comme temps totale de préparation de l'ensemble des tâches. Finalement, on obtient $C_{max} = n.S + p$.

Si le temps de préparation est inférieur au temps de traitement ($S < p$), alors les tâches seront traitées comme dans le cas où $k = 1$ avec un décalage de S unités de temps sur la deuxième machine (la préparation de la deuxième tâche commence à $t = S$ sur la deuxième machine). On obtient donc, $C_{max} = \frac{n(p+S)}{2} + S$ dans le cas ou n est pair et $C_{max} = \frac{(n+1)(p+S)}{2}$ dans le cas où n est impair (la deuxième machine sera libre pendant p unités de temps à la fin de l'ordonnancement). □

5 Méthodes approchés

Dans cette partie, nous proposons six heuristiques pour résoudre le problème considéré. Toutes ces heuristiques ont un point en commun qui est une procédure A1 dont le rôle est d'ordonnancer les tâches préalablement rangées.

La procédure $A1$ se déroule de la manière suivante : nous supposons qu'à l'instant $t = 0$ tous les ouvriers et toutes les machines sont disponibles, et tant que l'ensemble des tâches n'est pas encore vide, on sélectionne une tâche T_i de cet ensemble et on l'affecte au premier ouvrier libre pour la traiter sur la première machine disponible à l'instant $t = 0$. Ensuite, on élimine cette tâche de l'ensemble des tâches et la préparation et le traitement de la tâche qui la suit commence à $t := \max\{\min\{O_l\}, \min\{C_j\}\}$.

L'objectif de cette procédure est donc de déterminer à quel instant, par quel ouvrier et sur quelle machine on lance la préparation et le traitement d'une tâche.

La procédure $A1$ s'écrit alors :

Procédure A1 ;
début
- $t := 0$; $O_l := 0$; $C_j := 0$;
- **tantque** $T \neq \emptyset$
  faire
  - Prendre la première tâche T_i de la liste et l'affecter au premier ouvrier libre (soit l) pour la traiter sur la première machine (soit M_j) disponible à l'instant t ;
  - $O_l := t + S_i$; $C_j := O_l + p_i$;
  - $T := T \setminus \{T_i\}$; $t := \max\{\min\{O_l\}, \min\{C_j\}\}$;

  fait ;
fin.

O_l représente la date de disponibilité de l'ouvrier l et C_j représente la date de disponibilité de la machine j

La complexité de cette procédure est de $O(n)$.

Description des heuristiques Hi : La première phase consiste à ranger les tâches selon un certain ordre, ensuite, faire appel à la procédure $A1$.

Algorithme Hi ;
début - Ranger les tâches suivant la règle **Ri** ;
- Appliquer la procédure $A1$;

fin.

Nous avons définis 6 règles de rangement :

R1 : Ranger les tâches selon l'ordre LPT des temps de traitement (notée LPT_{p_i})

R2 : Ranger les tâches selon l'ordre SPT des temps de traitement (notée SPT_{p_i})

R3 : Ranger les tâches dans l'ordre décroissant de leurs temps de préparation (notée LPT_{S_i})

R4 : Ranger les tâches dans l'ordre croissant de leurs temps de préparation (notée SPT_{S_i})

R5 : Ranger les tâches dans l'ordre décroissant de leurs temps de traitement et de préparation (notée $LPT_{S_i+p_i}$)

R6 : règle qui range les tâches dans l'ordre croissant de leurs temps de traitement et de préparation (notée $SPT_{S_i+p_i}$)

Toutes ces règles s'exécute en $O(nlogn)$.

6 Résultats expérimentaux

Afin de tester les heuristiques développées nous avons généré plusieurs instances possédant différentes caractéristiques. Par ailleurs, chaque instance est

caractérisée par un triplet (machines, tâches, ouvriers) tel que le nombre de machines $m \in \{2, 3, 5, 10\}$, le nombre de tâches $n \in \{10, 20, 50, 100, 1000\}$ et le nombre d'ouvriers à $k \in \{1, ..., m\}$.

En ce qui concerne les temps de traitement des tâches et les temps de préparation, ils sont générés aléatoirement suivant la loi uniforme en prenant plusieurs intervalles de temps.

Pour m, n et k fixés, nous avons généré et testé 100 instances différentes. Pour chaque combinaison de nombre de machines, nombre de tâches et nombre d'ouvriers on compte le nombre de fois où l'heuristique H donne de meilleurs solutions, aussi le nombre de fois où elle donne une solution égale à la borne inférieure. La table 1 illustre le résumé de ces tests.

Tab. 1. Résumé des tests

Tests		H1	H2	H3	H4	H5	H6
$p_i \in [1, 10]$	H. meilleure	5138	349	794	556	7515	106
$S_i \in [1, 5]$	$C_{max} = \overline{M}$	1111	240	580	315	2167	106
$p_i \in [1, 20]$	H. meilleure	4745	278	762	534	7101	100
$S_i \in [1, 10]$	$C_{max} = \overline{M}$	1001	207	537	262	1761	100
$p_i \in [1, 50]$	H. meilleure	4764	210	687	540	6806	100
$S_i \in [1, 20]$	$C_{max} = \overline{M}$	878	167	462	250	1446	100
$p_i \in [1, 100]$	H. meilleure	4548	233	726	441	6666	100
$S_i \in [1, 50]$	$C_{max} = \overline{M}$	807	177	459	196	1331	100

D'après cette expérimentation, il s'avère que pour toutes les instances, les heuristiques $H1$ et $H5$ donnent de meilleures solutions. De plus, l'heuristique $H5$ est généralement meilleure que l'heuristique $H1$. En effet, sur 10000 instances générées (selon la loi uniforme) avec les p_i et les S_i uniformement distribués dans [1, 10] et [1, 5] respectivement, l'heuristique $H5$ donne une meilleure solution dans 7515 cas et une solution optimale dans 2167 cas.

En ce qui concerne la deuxième série des tests où les $p_i \in [1, 20]$ et $S_i \in [1, 10]$, l'heuristique $H5$ est meilleure dans 7101 cas contre 4745 cas pour $H1$. $H5$ donne la solution optimale dans 1761 cas contre 1001 cas pour $H1$.

Pour la troisième série des tests où $p_i \in [1, 50]$ et $S_i \in [1, 20]$, l'heuristique $H5$ est optimale dans 1446 cas sur 6806. Et pour la dernière série de tests il y a 1331 cas où elle donne des solutions optimales sur les 6666 cas où elle est meilleure.

Finalement, d'après cette expérimentation, il apparaît que l'heuristique $H5$ qui range les tâches selon la règle $LPT_{S_i+p_i}$ est très intéressante que les autres, de plus elle apporte la solution en moins de 16 ms pour les problèmes de petite taille et en moins de 141 ms pour les problèmes de grande taille. Pour une meilleure lisibilité, la figure 1 transcrit quelques résultats trouvés par l'heuristique $H5$.

Pour évaluer les six heuristiques, nous avons utilisé l'indicateur de performance de chaque heuristique par rapport à la borne inférieure. Ce dernier appelé aussi déviation et est donné par le quotient $\frac{(H-BI)}{BI}$ avec H la valeur du critère

donnée par l'heuristique et BI la borne inférieure. Pour avoir des résultats exploitables, on génère aléatoirement 100 instances et on calcule la moyenne de $\frac{(H-BI)}{BI}$.

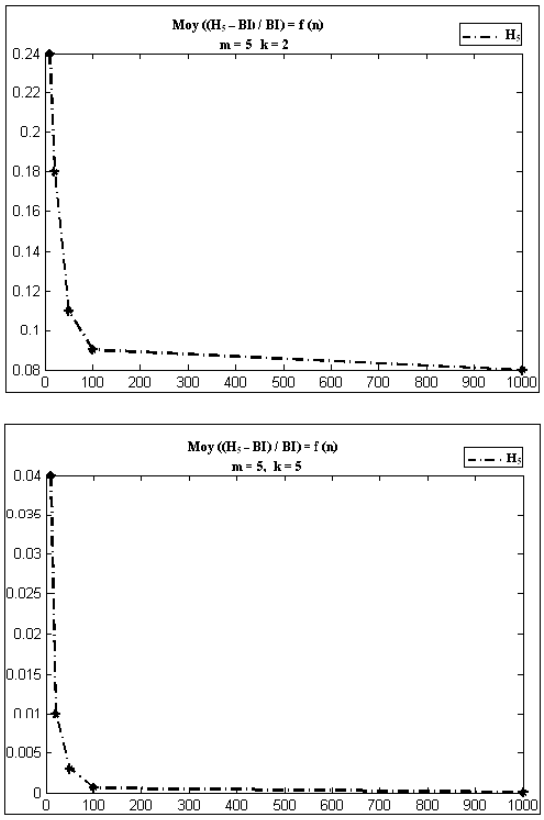

Fig. 1. Comportement de l'heuristique H5 par rapport à la borne inférieure pour $p_i \in [1, 10]$ et $S_i \in [1, 5]$

7 Conclusion

Dans cet article, nous avons traité un problème d'ordonnancement qui se pose dans un environnement à machines parallèles identiques en tenant compte des temps de préparation des tâches. L'objectif a été de déterminer des méthodes de résolution pour ce problème.

Nous avons, au cours de cet article, présenté une modélisation mathématique pour le problème et nous avons conclu que le problème général est NP-difficile.

Une borne inférieure, ainsi qu'un sous problème polynomial ont été proposés, comme nous avons développés six heuristiques.

Finalement, différents tests expérimentaux ont été réalisé sur des instances générées aléatoirement selon la loi uniforme. Après évaluation de ces heuristiques par rapport au nombre de fois où une heuristique l'emporte sur les autres, et le nombre de fois où la solution trouvée est égale à la borne inférieure, nous concluons que l'heuristique basée sur le rangement des tâches selon la règle $LPT_{S_i+p_i}$ est assez performante.

Le rapport de performance $\frac{4}{3} - \frac{1}{3m}$ de Graham pour $P2//C_{max}$ (règle LPT) reste valable pour le problème $Pm/S_i, k/C_{max}$ pour $k \geq m$ (règle $LPT_{S_i+p_i}$).

Dans les perspectives de nos études nous projetons d'étudier quelques extensions de ce problème, nous envisageons aussi d'améliorer les heuristiques et de résoudre le problème avec d'autres approches.

Références

1. Abdelkhodaee, AH., Wirth, A. : Scheduling parallel machines with a single server : some solvable cases and heuristics, Computers and Operations Research 29(3), 295-315 (2002).
2. Bettayeb, B., Kacem, I., Adjallah, K.H. : Ordonnancement sur machines parallèles identiques avec temps de préparation par famille : Application à la gestion des tâches de maintenance préventive. Université de Technologie de Troyes, $6^{ième}$ conférence froncophone de Modélisation et Simulation - MOSIM'06 (2006).
3. Boustta, M. : Minimisation du temps de fabrication sur une machine à injection avec réglages multiples, Thèse, University Laval. Québec (2003).
4. Cochran, J., Horng, SM., Fowler, J. : A multi- population genetic algorithm to solve multi-objective scheduling problems for parallel machines. Computer and Operations Research 30, 1087-1102 (2003).
5. Dell'Amico, M., Iori, M., Martello, S. : Heuristic Algorithms and Scatter Search for the Cardinality Constrained P//Cmax problem, journal of heuristics 10, 169-204 (2004).
6. Ghosh Jay, B. : Batch scheduling to minimize total completion time. Operations Research Letters 16, 271-275 (1994).
7. Karp, R. : Complexity of computer computations. R.E. Miller, J.W. Thatcher (eds.), Plenum Press, New-York (1972).
8. Koulamas, CP. : Scheduling two parallel semiautomatic machines to minimize machine interference. Computers and Operations Research 23(10), 945-956 (1996).
9. Kravchenko, SA. Werner, F. : Parallel machine scheduling problem with a single server. Mathematical and Computer Modelling 26(12), 1-11 (1997).
10. Mellouli, R., Sadfi, C., Kacem, I., Chu, C. : Ordonnancement sur machine parallèles avec contraintes d'indiponibilité. Université de Technologie de Troyes, $6^{ième}$ conférence froncophone de Modélisation et Simulation - MOSIM'06 (2006).
11. Zouba, M., Baptiste, P., Rebaine, D. : Ordonnancement de jobs sur deux machines parallèles en presence d'un seul opérateur, $6^{ième}$ conférence froncophone de Modélisation et Simulation - MOSIM'06 (2006).