RAIRO. RECHERCHE OPÉRATIONNELLE

WENXUE HAN
PIERRE DEJAX

Une heuristique pour le problème d'ordonnancement de type $n/m//F/C_{\max}$ avec la présence de machines goulots

RAIRO. Recherche opérationnelle, tome 24, n° 4 (1990), p. 315-330

<http://www.numdam.org/item?id=RO_1990__24_4_315_0>

Recherche opérationnelle/Operations Research
(vol. 24, n° 4, 1990, p. 315 à 330)

UNE HEURISTIQUE POUR LE PROBLÈME D'ORDONNANCEMENT DE TYPE $n/m//F/C_{max}$ AVEC LA PRÉSENCE DE MACHINES GOULOTS (*)

par Wenxue HAN (1) et Pierre DEJAX (1)

Résumé. – *Nous présentons une heuristique pour le problème d'ordonnancement de type $n/m//F/C_{max}$. Cette heuristique est basée sur l'identification de la machine goulot. Les résultats des expériences numériques montrent que cette approche conduit à une excellente performance lors de présence de machines goulots.*

Mots clés : Heuristique; machine goulot; flow-shop; problème d'ordonnancement.

Abstract. – *We present a heuristic algorithm for the flow-shop scheduling problem of the $n/m//F/C_{max}$ in order to minimize the makespan. The algorithm is based on the identification of the bottleneck machine. Experimental results show that the performance of the algorithm is better than existing methods with the presence of the bottleneck machine.*

Keywords : Heuristic; bottleneck machine; flow-shop; scheduling problem.

1. INTRODUCTION

Nous nous intéressons dans cet article au problème d'ordonnancement de type $n/m//F/C_{max}$ où le critère d'optimisation est de minimiser la date d'achèvement de la dernière tâche sur la dernière machine (Min C_{max}). Il s'agit d'ordonnancer n tâches sur m machines où toutes les tâches doivent suivre le même ordre de fabrication sur l'ensemble des moyens de production (flow-shop). La préemption n'est pas autorisée. Le problème est dit statique en ce sens que toutes les tâches peuvent débuter à la date zéro. Les temps

(*) Reçu en juin 1990.
(1) École centrale Paris-Leis, 92295 Châtenay-Malabry Cedex, France.

Recherche opérationnelle/Operations Research, 0399-0559/90/04 315 16/$ 3.60

nécessaires d'exécution de chacune d'elles sur chacune des machines sont des données du problème.

Nous consacrons la deuxième partie à présenter un état de l'art concernant notamment les algorithmes existant pour résoudre ce type de problème. Dans la troisième partie, nous proposons une nouvelle heuristique, baptisée PHD (Procédure Han Dejax). Cette heuristique est particulièrement appropriée aux problèmes présentant des machines goulot. La quatrième partie contient les résultats d'expériences numériques permettant de comparer les performances de cette heuristique à celles de l'heuristique de Campbell, Dudek et Smith, qui est considérée comme l'une des meilleures connues jusqu'à présent. Enfin, la dernière partie contient nos conclusions et nos suggestions pour des recherches futures.

2. ÉTAT DE L'ART CONCERNANT LE PROBLÈME D'ORDONNANCEMENT DE TYPE $n/m//F/C_{\max}$

Le problème d'ordonnancement de type $n/m//F/C_{\max}$ est un problème NP-complet d'un grand intérêt pratique, qui a donné lieu à de nombreuses recherches. Seules les méthodes d'énumération contrôlée peuvent aboutir à la solution optimale à coup sûr dans le cas général. En fait, seul Baker (1974) présente une formulation relevant de la programmation en nombres entiers. Cette formulation bien que fort économique en nombre de variables n'est utilisable que sur de très petits problèmes. Les heuristiques paraissent être la seule méthode efficace pour résoudre les problèmes de grande taille, pour lesquels nous ne connaissons pas encore de solution optimale.

2.1. Algorithme optimal de Johnson (1954)

Un seul algorithme optimal proposé par Johnson (1954) existe pour résoudre le problème d'ordonnancement considéré dans la littérature. Il se limite au problème d'ordonnancement à deux machines et consiste à :

– rechercher la tâche i, $i=1, \ldots, n$, dont le temps d'exécution t_{ij} (pour la machine j, $j=1,2$) est le plus faible possible ;

– si $j=1$, placer cette tâche à la première place disponible en début de la séquence d'ordonnancement, et si $j=2$, placer cette tâche à la dernière place disponible ;

– supprimer la tâche i des tâches restant à programmer ; s'il reste plus d'une tâche à programmer, revenir à la première étape ; s'il n'en reste qu'une, sa position est imposée puisqu'il ne reste plus qu'une seule place à prendre.

L'algorithme proposé par Johnson a été à la base de nombreux travaux ultérieurs sur le problème d'ordonnancement qui nous intéresse ici (Proust, 1989).

2.2. Méthodes heuristiques

Les travaux concernant des heuristiques pour résoudre les problèmes de type $n/m//F/C_{\max}$ sont nombreux et paraissent régulièrement dans la littérature spécialisée. Page (1961), Palmer (1965), Petrov (1966), Campbell, Dudek et Smith (1970), Gupta (1971) et (1976), Dannenbring (1977), King et Spachis (1980), Chen (1983), Nawaz, Enscore et Ham (1983), Park, Pegden et Enscore (1984), Hundal et Rajgopal (1988) jalonnent presque 30 ans d'intérêt pour ce problème. On connaît l'importance de l'heuristique CDS, référence en la matière, mais que Dannenbring puis King et Spachis améliorèrent avec leurs procédures RAES et LWBJD. Il n'existe cependant pas de comparaison générale de ces différentes méthodes. Nous rappelons ici les principaux travaux existants.

2.2.1. *Heuristique de Palmer* (1965)

Historiquement, une des premières heuristiques développées l'a été par Palmer (1965). Son raisonnement est du type dynamique. Il s'agit d'une extension de la règle SPT (Shortest Processing Time), dans la mesure où ici, les tâches sont classées dans l'ordre croissant de leur tendance à avoir des temps opératoires plus longs sur la fin de la séquence d'opération. On calcule alors un index de pente :

$$f(i) = \sum_{j=1}^{m} (m - 2j + 1)\, t_{ij} \qquad \text{pour} \quad i = 1 \ldots n,$$

et on classe les tâches i en ordre croissant de $f(i)$.

2.2.2. *Première heuristique de Gupta* (1971)

Gupta (1971), s'appuyant sur la règle de Johnson, propose un autre index de pente :

$$f(i) - \frac{\text{signe}\,(t_{i1} - t_{im})}{\min\limits_{1 \leqq j \leqq m-1} (t_{ij} + t_{ij+1})} \qquad \text{pour} \quad i = 1 \ldots n,$$

et

$$\text{signe}\,(t_{i1} - t_{im}) = \begin{cases} 1 & \text{si} \quad t_{i1} - t_{im} \geqq 0 \\ -1 & \text{sinon} \end{cases}.$$

Les tâches sont classées en ordre croissant de $f(i)$. En cas d'égalité on préfère celle ayant la plus faible somme des temps opératoires. Les deux heuristiques de Palmer et Gupta n'ont pas été, à notre connaissance, comparées entre elles, parce qu'entre temps un autre type d'heuristique (CDS) a été développé.

2.2.3. *Heuristique CDS de Campbell, Dudek et Smith* (1970)

Cette méthode s'appuie sur la règle de Johnson. Elle consiste à générer $m-1$ solutions en appliquant l'algorithme de Johnson sur deux machines fictives. La première regroupe les k premières machines, la deuxième regroupe les k dernières. k varie de 1 à $m-1$. Les temps opératoires de chaque tâche i sur ces deux machines fictives sont la somme des temps opératoires sur les machines qu'ils regroupent. Nous pouvons les définir ainsi :

$$P_{i1}^{k} = \sum_{j=1}^{k} t_{ij}, \qquad P_{i2}^{k} = \sum_{j=m-k+1}^{m} t_{ij} \qquad \text{pour} \quad i = 1 \ldots n.$$

La meilleure des $m-1$ solutions, au sens de la plus faible date d'achèvement de la dernière tâche sur la dernière machine pour chaque séquence proposée, est retenue comme la solution du problème à m machines.

2.2.4. *Deuxième heuristique de Gupta* (1976)

Page (1961) démontre que l'algorithme de Johnson peut être formulé de la façon suivante si nous classons les tâches i en ordre croissant de la fonction $f(i)$, avec :

$$f(i) = \frac{\text{signe}\,(t_{i1} - t_{i2})}{\min\,(t_{i1}, t_{i2})} \qquad \text{pour} \quad i = 1 \ldots n,$$

et

$$\text{signe}\,(t_{i1} - t_{i2}) = \begin{cases} 1 & \text{si} \quad t_{i1} - t_{i2} \geqq 0 \\ -1 & \text{sinon.} \end{cases}$$

En utilisant cette formulation, Gupta propose (1976) une heuristique similaire à CDS, mais ne différant que par la manière d'arbitrer les conflits ; ici l'ordre choisi entre deux tâches en cas de conflit est celui des séquences ultérieures, les séquences antérieures n'entrant en ligne de compte que lorsque les ultérieures ne suffisent pas.

Gupta démontre que l'erreur moyenne par rapport à la solution optimale est légèrement plus faible qu'avec CDS. Cette remarque reste valable quelque soit la taille du problème.

2.2.5. *Heuristiques de Dannenbring* (1977)

Dannenbring (1977) a montré qu'en partant d'une « bonne solution » et en lui appliquant une technique d'amélioration du type génération des solutions proches, et ceci de façon itérative, on retient généralement de solutions bien meilleures.

La « bonne solution » initiale est créée par une généralisation de l'algorithme de Johnson ; le calcul des temps opératoires pour les deux machines fictives est fait de la manière suivante :

$$P_{i1} = \sum_{j=1}^{m} (m-j+1)\, t_{ij}, \qquad P_{i2} = \sum_{j=1}^{m} j t_{ij}.$$

On applique alors l'algorithme de Johnson sur ces deux machines fictives 1 et 2 qui tiennent compte du fait qu'un retard est plus important pour les opérations de la fin des tâches que pour les premières (ceci étant lié au choix du critère C_{max}). L'amélioration de cette solution donne lieu à deux heuristiques :

RACS : $(n-1)$ solutions sont générées à partir de la solution initiale par transposition d'une paire de tâches adjacentes à chacune d'elles. Le calcul de C_{max} pour chacune de ces solutions permet de déterminer la meilleure.

RAES : la procédure commence par l'application de RACS, puis la solution la meilleure est prise comme solution initiale pour réappliquer RACS. Ce processus itératif est répété jusqu'à ce qu'on n'obtienne plus d'amélioration.

Les essais faits par l'auteur montrent que RAES est une heuristique donnant des résultats meilleurs que toutes celles décrites jusqu'ici (y compris une technique consistant à s'arrêter à la première solution possible générée par la méthode de séparation et évaluation progressives. Ceci est valable quelle que soit la taille des problèmes en nombre de machines ou de tâches. RACS semble meilleure que CDS dans le cas des problèmes ne dépassant pas 6 tâches et 10 machines.

Si les temps de calcul des méthodes RACS et CDS sont comparables, la performance de RAES est fortement contrebalancée par l'augmentation des temps de calcul, qui sont environ 20 fois plus longs que pour CDS ou RACS, bien qu'ils soient quand même sept fois plus rapide que pour la méthode de séparation et évaluation progressives citée plus haut.

Il faut noter que le temps de calcul pour l'heuristique RAES n'augmente réellement qu'avec le nombre de tâches et que c'est pour un nombre de machines élevé que la différence de temps de calcul entre CDS et RAES est la plus significative.

2.2.6. *Heuristiques de King et Spachis* (1980)

En partant du principe que minimiser le temps total d'achèvement dans le cas du « flow-shop » est équivalent à minimiser la somme des délais introduits en cours de production, King et Spachis (1980) ont proposé de nouvelles heuristiques pour lesquelles trois types de délais sont introduits :

– le délai de lancement : temps écoulé avant le lancement de la première opération sur la machine ;

– le délai inter-tâches : temps écoulé entre deux opérations sur une machine. Il regroupe (de manière non exclusive) :

– le délai de type « arrière » qui peut être réduit en avançant le début de l'opération suivante,

– le délai de type « avant » qui ne peut être réduit de cette manière ;

– le délai de sortie : écart entre le temps total d'achèvement et la date d'achèvement de la dernière opération sur la machine considérée.

Cinq heuristiques sont proposées ; elles produisent toutes autant de solutions que de tâches puisque la première est choisie arbitrairement et que l'on peut donc générer une solution pour chaque tâche initiale. Il s'agit d'agréger à la solution partielle la tâche produisant :

1. Le plus petit total de délais avant : (LFD).
2. Le plus petit total de délais arrière : (LBD).
3. Le plus petit total de délais inter-tâches : (LBJD).
4. Le plus petit total pondéré de délais inter-tâches : (LWBJD).
5. Le plus grand total d'avancement possible d'opérations : (MLSS).

La pondération du délai inter-tâches (LWBJD) se fait avec un coefficient égal au numéro de la machine afin de tenir compte de la plus grande importance d'un retard éventuel sur les dernières machines que sur les premières.

Les essais de ces heuristiques faits par les auteurs utilisent les mêmes critères de mesure que ceux rencontrés en général dans ce genre de problèmes (les heuristiques de type « flow-shop ») : pourcentage d'erreur par rapport à la solution optimale, rang moyen de l'heuristique sur les problèmes testés... Par contre les essais n'ont été effectués que sur des problèmes d'assez faible dimension (35 tâches et 5 machines au maximum).

Dans ce cadre, il ressort de façon très nette la supériorité de l'heuristique LWBJD (toujours pour les petits problèmes) qui, seule, dépasse l'heuristique CDS qui sert généralement de référence. Les heuristiques LBJD LFD donnent

également de bons résultats, puis les autres sont équivalentes, en performance, au classement au hasard.

Un autre avantage important de l'heuristique LWBJD est sa facilité d'application puisqu'elle ne nécessite que très peu de temps de calcul et peut être utilisée dans un contexte dynamique (à condition de considérer le processus comme continu, auquel cas il n'y a pas à proprement parler de tâche initiale et donc une seule solution générée à chaque fois). Les performances de cette heuristique n'ont cependant pas été évaluées dans le cadre de problèmes de plus grande dimension.

2.2.7. *Heuristique de Ham*

Ham propose une heuristique qui consiste tout d'abord à décomposer la matrice initiale des temps opératoires en deux sous-matrices. Basées sur cette décomposition, deux solutions sont générées pour chaque problème dont la meilleure est retenue comme la solution finale. Elle procède de la manière suivante :

– la première matrice regroupe les premières $m/2$ (si m est pair) ou $(m+1)/2$ machines (sinon) ; la deuxième regroupe les dernières $m/2$ (si m est pair) ou $(m+1)/2$ machines (sinon) ;

– on calcule la somme des temps opératoires pour chaque sous-matrice et pour chaque tâche i :

$$P_{i1} = \sum_{j=1}^{m/2} t_{ij} \quad \text{ou} \quad P_{i1} = \sum_{j=1}^{(m+1)/2} t_{ij},$$

$$P_{i2} = \sum_{j=(m+2)/2}^{m} t_{ij} \quad \text{ou} \quad P_{i2} = \sum_{j=(m+1)/2}^{m} t_{ij};$$

– on calcule $P_{i2} - P_{i1}$ pour chaque tâche i, $i = 1, \ldots, n$;

– on établit un premier ordonnancement dans l'ordre décroissant de $(P_{i2} - P_{i1})$;

– on établit un deuxième ordonnancement en classant les tâches qui ont $(P_{i2} - P_{i1})$ positif dans l'ordre croissant de P_{i1}, et les tâches qui ont $(P_{i2} - P_{i1})$ négatif dans l'ordre décroissant de P_{i2} ;

– on retient celui parmi les deux qui conduit au plus petit C_{max} comme ordonnancement.

2.2.8. *Heuristique de Chen* (1983)

Chen (1983) propose une heuristique qui consiste à procéder de la manière suivante :

– on calcule la somme de temps opératoires $S(i)$ pour chaque tâche i avec :

$$S(i)=\sum_{j=1}^{m} t_{ij}, \qquad \forall_i=1 \ldots n;$$

– on trouve la tâche c qui a la $S(i)$ la plus importante et on enlève cette tâche de l'ensemble de tâches non ordonnancées ;

– on ordonnance les tâches qui ont $P_{i1} \leqq P_{im}$ dans l'ordre croissant de P_{i1} et on obtient un ordonnancement partiel S_A ;

– on ordonnance les tâches qui ont $P_{i1} > P_{im}$ dans l'ordre décroissant de P_{im} et on obtient un ordonnancement partiel S_B ;

– on retient comme solution finale l'ordonnancement (S_A, c, S_B).

2.2.9. *Heuristique NEH de Nawaz, Enscore et Ham* (1983)

Nawaz, Enscore et Ham (1983) proposent une heuristique qui consiste à trouver une solution initiale puis à l'améliorer par $n(n+1)/2-1$ permutations. Nous l'indiquons ci-dessous :

– on calcule la somme de temps opératoires $S(i)$ pour chaque tâche i avec :

$$S(i)=\sum_{j=1}^{m} t_{ij}, \qquad \forall_i=1 \ldots n;$$

et on classe les tâches dans l'ordre décroissant de $S(i)$. La solution initiale est ainsi trouvée ;

– on considère les deux premières tâches ayant les temps opératoires totaux les plus grands et les ordonne l'un par rapport à l'autre en testant les valeurs de C_{max} des deux façons possibles. On retient la séquence qui a le C_{max} minimal, par exemple (T_j, T_i). Puis on choisit la troisième tâche possédant le plus grand temps opératoire total parmi les $(n-2)$ restants et on la place dans toutes les positions possibles dans la séquence obtenue à l'étape précédente. On retient la meilleure parmi les trois. Ce processus de placement se répète jusqu'à ce que toutes les tâches aient été placées.

2.2.10. *Heuristique PPE de Park, Pegden et Enscore* (1984)

Park, Pegden et Enscore (1984) proposent une heuristique qui semble très efficace. Elle consiste à générer $m-1$ solutions en appliquant l'algorithme de Johnson sur deux machines fictives. La première regroupe les k premières machines, la deuxième regroupe les k dernières. k varie de 1 à $m-1$. Les temps opératoires sur ces deux machines fictives sont définis ainsi :

$$P^k_{i1} = \sum_{j=1}^{k} (m-j+1)\, t_{ij},$$

et $\forall_i = 1 \ldots n, \qquad \forall_k = 1 \ldots m-1.$

$$P^k_{i2} = \sum_{j=1}^{k} (m-j+1)\, t_{im-j+1}$$

La meilleure des $m-1$ solutions au sens de la plus faible date d'achèvement de la dernière tâche sur la dernière machine pour chaque séquence proposée, est retenue comme la solution du problème à m machines.

2.2.11. *Heuristique de Hundal et Rajgopal* (1988)

Hundal et Rajgopal (1988) proposent une heuristique qui consiste à procéder de façon suivante :

– on établit un premier ordonnancement, grâce à l'heuristique de PALMER ;

– on bâtit une pseudo-machine avec les temps opératoires fictifs suivants :

$$t'_i = \sum_{j=1}^{m} (m-2j)\, t_{ij}, \qquad \forall_i = 1 \ldots n ;$$

– on classe les tâches dans l'ordre des t'_i décroissants et on obtient un deuxième ordonnancement ;

– on bâtit une autre pseudo-machine avec :

$$t''_i = \sum_{j=1}^{m} (m-2j+2)\, t_{ij}, \qquad \forall_i = 1 \ldots n ;$$

on classe les tâches dans l'ordre des t''_i décroissants et on obtient une troisième séquence possible ;

– la séquence qui, parmi les trois, conduit au plus petit C_{max} est l'ordonnancement retenu.

3. DESCRIPTION DE L'HEURISTIQUE PHD (Procédure Han et Dejax)

Nous proposons une nouvelle démarche basée sur la notion de machine goulot pour résoudre le problème d'ordonnancement de type $n/m//F/C_{\max}$. Il est considéré comme un problème de gestion des flux de tâches. Ce qui revient à dire que minimiser la date d'achèvement est équivalent à faire passer le plus vite possible le flux dans le système avec respect des contraintes imposées. C'est la machine goulot, autrement dit celle la plus chargée, qui détermine la vitesse du flux. Il faut donc que la machine goulot commence à fonctionner le plus tôt possible et que les temps des travaux restant à faire sur les machines en aval soient minimisés une fois les travaux sur la machine goulot terminés. Il faut également éviter l'arrêt de la machine goulot en cours de fonctionnement du fait du manque de travaux juste devant elle.

En partant de la machine goulot, nous décomposons l'ensemble des machines réelles en deux machines fictives 1 et 2. La première regroupe les machines qui se trouvent en amont de la machine goulot; la deuxième regroupe celles en aval. La machine goulot ne se présente dans aucun groupe. Dans le cas où la machine goulot est la première (respectivement la dernière), elle est considérée comme la première (respectivement la dernière) machine fictive. On détermine l'ordonnancement optimal des tâches sur les deux machines fictives grâce à l'algorithme de Johnson (1954). On répète ce calcul en considérant successivement comme machine goulot les machines classées dans l'ordre décroissant de leur charge. On retient comme solution finale l'ordonnancement qui conduit à la plus petite date d'achèvement $C_{\max}$. Nous décrivons formellement l'agorithme ci-après :

Étape 1 : Calcul de la charge de chaque machine.

Elle est définie par

$$W_j = \sum_{i=1}^{n} t_{ij}, \qquad j = 1, \ldots, m.$$

Soit J l'ensemble des machines j, $j = 1, \ldots, m$.

Étape 2 : Choix d'une machine goulot.

Sélectionner la machine k, telle que

$$W_k = \operatorname*{Max}_{k \in J} \; \operatorname*{Max}_{j \in J} W_j.$$

Étape 3 : Calcul des temps opératoires des deux machines fictives.

Soit P_{i1} et P_{i2} les temps opératoires des tâches i, $i=1, \ldots, n$, sur les machines fictives 1 et 2.

si $k=1$, alors : $P_{i1}=t_{i1}$;

si $k=m$, alors : $P_{i2}=t_{im}$;

sinon,

$$P_{i1}=\frac{1}{k-1}\sum_{j=1}^{k-1} t_{ij}, \qquad P_{i2}=\frac{1}{m-k}\sum_{j=k+1}^{m} t_{ij}.$$

Étape 4 : Calcul de la fonction de classement par

$$f(i)=\frac{\text{signe}\,(P_{i1}-P_{i2})}{\text{Min}\,(P_{i1}, P_{i2})}$$

avec :

$$\text{signe}\,(P_{i1}-P_{i2})=\begin{cases} 1 & \text{si} \quad P_{i1}-P_{i2}\geqq 0 \\ -1 & \text{sinon} \end{cases}.$$

Étape 5 : Détermination de l'ordonnancement.

Il correspond au classement des tâches en ordre croissant en fonction de $f(i)$, $i=1, \ldots, n$. La date d'achèvement est $C^k_{\max}$.

Étape 6 : Mise à jour.

Enlever k de l'ensemble J. Si $J=\varnothing$, aller à l'étape 7, sinon, aller en 2.

Étape 7 : Choix du meilleur ordonnancement.

Retenir comme solution l'ordonnancement correspondant à la machine goulot k telle que

$$C^k_{\max}\leqq C^j_{\max}, \qquad j=1, \ldots, m.$$

4. EXPÉRIENCES NUMÉRIQUES

Afin de valider cette approche, nous avons effectué une comparaison entre l'heuristique PHD et l'heuristique CDS avec un grand nombre de tests. Les temps opératoires sont générés de façon aléatoire suivant une loi log-normale comme suggéré par Asselin de Beauville (1974) et pratiqué par la plupart des auteurs.

L'heuristique PHD est proposée pour résoudre le problème d'ordonnancement dans le cas de présence de machines goulots. Afin d'évaluer sa performance même dans le cas où il n'y a pas de présence de machines goulots, nous avons retenu un scénario. Des problèmes de taille $n \times m$ sont étudiés

avec $n = \{ 6, 10, 15 \}$ et $m = \{ 5, 7, 10 \}$. Pour chaque taille nous avons généré 50 jeux d'essais, donc au total 450 jeux.

Dans la deuxième série de tests, un coefficient comportant la valeur 1.5, 2, 2.5 ou 3 est choisi par hasard et accordé à une colonne quelconque pour chaque problème de manière à faire paraître un goulot dans le système. Nous avons retenu trois scénarios pour lesquels les valeurs de l'espérance et de la variance sont (20, 25), (30, 35), (40, 25). Les problèmes de taille $n \times m$ sont étudiés pour chaque scénario avec $n = \{ (10, 15, 20); (30, 40, 50) \}$ et $m = \{ (7, 10, 15, 20); (25, 30) \}$. Pour chaque taille nous avons généré 50 jeux d'essais, donc au total 2 700 jeux. On notera $\mathrm{CDS}C_{\max}$ et $\mathrm{PHD}C_{\max}$ comme les résultats obtenus par les algorithmes CDS et PHD. Pour un jeu de donnée, l'indicateur de comparaison entre les deux algorithmes est défini de la façon suivante :

$$\mathrm{Ind}_i = 100 \times \frac{\mathrm{CDS}C_{\max} - \mathrm{PHD}C_{\max}}{\min(\mathrm{CDS}C_{\max}, \mathrm{PHD}C_{\max})}.$$

Nous présentons ci-après une comparaison des résultats obtenus par les algorithmes PHD et CDS. Les tableaux de résultats se lisent de la façon suivante : Dans les deux premières colonnes, chaque ligne correspond à un nombre de machines (m) et de tâches (n). Chacune des cinq colonnes suivantes correspond à l'intervalle dans lequel s'est situé l'indicateur de comparaison des deux algorithmes. Dans chaque ligne, on a indiqué le nombre de problèmes ayant conduit à un indicateur dans l'intervalle correspondant à chacune des colonnes.

4.1. Tests sans machine goulot

Pour ce scénario, l'espérance est fixée à 20 et la variance à 40. Sur 450 problèmes résolus : 250, soit 55,56 % le sont mieux par CDS que par PHD; 92 problèmes, donc 20,44 %, sont résolus de façon identique par les deux heuristiques; 108 problèmes, donc 24 %, conduisent à de meilleurs résultats avec PHD qu'avec CDS. Les résultats sont indiqués dans le tableau I.

4.2. Tests avec machine goulot

Les problèmes fréquemment rencontrés dans la pratique ne comporte qu'une machine à charge significativement supérieure aux autres. Pour cette raison nous avons jugé suffisant de limiter les itérations de l'algorithme au nombre de machines goulots ayant une charge supérieure à la moyenne des machines.

TABLEAU I

m	n	(- ∞,-2)	[-2, 0)	0	(0, 2)	(2,∞)
	6	4	10	27	5	4
5	10	8	16	8	12	6
	15	10	15	12	8	5
	6	8	18	14	8	2
7	10	12	18	8	7	5
	15	12	15	9	10	4
	6	5	21	10	10	4
10	10	13	24	3	8	2
	15	14	27	1	8	0
SUM:	450	86	164	92	76	32

Pour le premier scénario, l'espérance est fixée à 20 et la variance à 25. Sur 900 problèmes résolus : 121 le sont mieux par CDS que par PHD; 321 problèmes, donc 35,67 %, sont résolus de façon identique par les deux heuristiques; 458 problèmes, donc 50,89 %, conduisent à de meilleurs résultats avec PHD qu'avec CDS. Les résultats sont indiqués dans le tableau II.

TABLEAU II

m	n	(- ∞,-2)	[-2, 0)	0	(0, 2)	(2,∞)
	10	1	8	8	30	3
7	15	0	2	5	39	4
	20	0	5	4	40	1
	10	0	3	38	9	0
10	15	0	5	30	15	0
	20	0	0	31	19	0
	10	1	4	32	12	1
15	15	0	1	28	21	0
	20	0	1	30	19	0
	10	0	6	26	18	0
20	15	0	8	23	19	0
	20	0	2	25	22	1
	30	0	12	8	30	0
25	40	0	17	4	21	8
	50	0	14	3	30	3
	30	0	10	14	26	0
30	40	3	14	3	20	10
	50	0	4	9	34	3
SUM:	900	5	116	321	424	34

Pour le deuxième scénario, l'espérance est fixée à 30 et la variance à 35. Sur 900 problèmes résolus : 103 le sont mieux par CDS que par PHD; 313 problèmes, donc 34,78 %, sont résolus de façon identique par les deux heuristiques; 484 problèmes, donc 53,78 %, démontrent une démarche plus

avantageuse de PHD par rapport à CDS. Les résultats sont indiqués dans le tableau III.

TABLEAU III

m	n	$(-\infty,-2)$	$[-2, 0)$	0	$(0, 2)$	$(2,\infty)$
	10	0	7	8	35	0
7	15	1	4	3	42	0
	20	0	5	6	39	0
	10	0	4	35	11	0
10	15	0	1	36	13	0
	20	0	3	37	10	0
	10	1	11	26	11	1
15	15	0	0	28	22	0
	20	0	3	28	19	0
	10	0	6	27	17	0
20	15	0	5	20	25	0
	20	0	3	23	24	0
	30	0	13	8	29	0
25	40	0	5	7	38	0
	50	0	7	9	34	0
	30	0	10	8	32	0
30	40	0	8	4	37	1
	50	0	7	0	43	0
SUM:	900	2	101	313	482	2

Pour le troisième scénario, l'espérance est fixée à 40 et la variance à 25. Sur 900 problèmes résolus : 104 le sont mieux par CDS que par PHD; 321 problèmes, donc 35,67 %, sont résolus de façon identique par les deux heuristiques; 473 problèmes, donc 52,56 %, démontrent une démarche plus avantageuse de PHD par rapport à CDS. Les résultats sont indiqués dans le tableau IV.

On peut faire une synthèse de l'ensemble de ces passages. Dans le cas d'absence de machine goulot, sur 450 problèmes résolus, 250 problèmes, soit 55,56 % le sont mieux par CDS que par PHD. 92, soit 20,44 % sont résolus de façon identique par les deux heuristiques. Au total 200, donc 44,44 %, PHD s'est révélée aussi, ou plus avantageuse que CDS. Dans le cas où une machine goulot est présente, sur 2 700 problèmes résolus, 328 problèmes, soit 12,15 % le sont mieux par CDS que par PHD. 955, soit 35,37 % sont résolus de façon identique par les deux heuristiques. Au total 2 372, donc 87,85 %, PHD s'est révélée aussi, ou plus avantageuse que CDS.

5. CONCLUSION

Le problème d'ordonnancement de type flow-shop, à n tâches et m machines, où on veut minimiser la date d'achèvement de l'ordonnancement est rencontré dans de nombreuses applications industrielles. Nous ne connaissons

TABLEAU IV

m	n	(- ∞,-2)	[-2, 0)	0	(0, 2)	(2,∞)
	10	0	7	11	32	0
7	15	0	5	10	35	0
	20	0	5	5	40	0
	10	0	1	44	5	0
10	15	0	0	36	14	0
	20	0	2	36	12	0
	10	0	2	33	14	1
15	15	0	7	9	34	0
	20	0	9	4	37	0
	10	0	6	27	16	1
20	15	0	7	23	20	0
	20	0	8	19	23	0
	30	0	6	11	33	0
25	40	0	5	12	33	0
	50	0	6	11	33	0
	30	0	9	12	29	0
30	40	0	11	14	25	0
	50	0	8	4	38	0
SUM:	900	0	104	321	473	2

pas de solution optimale à ce problème bien que presque 30 ans de recherches y aient été consacrées. Nous avons présenté un état de l'art sur ce sujet et proposé une nouvelle approche basée sur la machine goulot. Les tests que nous avons effectués montrent que cette approche conduit à une excellente performance lors de présence de machines goulots, avec également un temps de calcul raisonnable. Il nous semble que cette approche pourrait apporter de bonnes solutions aux autres problèmes d'ordonnancement. Les recherches futures devaient porter sur l'application de cette approche aux autres problèmes d'ordonnancement et sur la méthode d'identification des goulots.

BIBLIOGRAPHIE

J. P. ASSELIN DE BEAUVILLE, Les sous-programmes de simulation statistique, *Revue de statistique appliquée*, 1974, *XXII*, n° 4, p. 57-87.

K. R. BAKER, Introduction to Sequencing and Scheduling, John Wiley and Sons, p. 169 et suivantes, 1974.

H. G. CAMPBELL, R. A. DUDEK et N. L. SMITH, A Heuristic Algorithm for the *n* Job, *m* Machine Sequencing Problem, *Management Science*, 1970, *16*, n° 10, p. 630-637.

R. Q. CHEN, Une nouvelle heuristique pour le problème d'ordonnancement en flow-shop, *Modernisation de management*, 1983, n° 1, p. 14-16.

DANNENBRING et G. DAVID, An Evaluation of Flowshop Sequencing Heuristics, *Management Science*, 1977, *23*, n° 11, p. 1174-1182.

J. N. D. GUPTA, A General Algorithm for the $n \times m$ Scheduling Problem, *Int. J. Prod. Res.*, 1969, *7*, p. 241.

J. N. D. GUPTA, A Functional Heuristic Algorithm for the Flowshop Scheduling Problem, *Operational Research Quarterly*, 1971, *22*, n° 1, p. 39-47.

J. N. D. GUPTA, A Heuristic Algorithm for the Flowshop Sequencing Problem, *Revue Française d'automatique, informatique et Recherche opérationnelle*, 1976, *10*, n° 6, p. 63-73.

T. S. HUNDAL et J. PAJGOPAL, An Extension of Palmer's Heuristic for the Flowshop Scheduling Problem, *Int. J. Prod. Res.*, 1988, *26*, (6), p. 1119-1124.

S. M. JOHNSON, Optimal Two- and Three-Stage Production Schedules with Setup Times Included, *Nav. Res. Log. Q.*, 1954, *1*, n° 1, p. 61-68.

S. M. JOHNSON, Discussion: Sequencing *n* Jobs on Two Machines with Arbitrary Time Lags, *Management Science*, 1959, *5*, n° 3, p. 299-303.

J. R. KING et A. S. SPACHIS, Heuristic for Flowshop Scheduling, *Int. J. Prod. Res.*, 1980, *18*, (3), p. 345-357.

M. NAWAZ, E. ENSCORE et L. HAM, A Heuristic Algorithm for the *m* Machine, *n* Job Flowshop Sequence Problem, *Omega*, 1983, *11*, p. 1.

E. S. PAGE, An Approach to the Scheduling of Jobs on Machines, *J. Roy. Statist. Soc.*, 1961, *23*, p. 484-492.

D. S. PALMER, Sequencing Jobs through a Multi-Stage Process in the Minimum Total Time- a quick Method of Obtaining a Near Optimum, *Opl. Res. Q.*, *16*, p. 101.

Y. PARK, C. PEGDEN et E. ENSCORE, A Survey and Evaluation of Static Flowshop Scheduling Heuristics, *Int. J. Prod. Res.*, 1984, *22*, n° 1, p. 127-141.

PENG SI OW, Focused Scheduling in Proportionate Flowshops, *Management Science*, 1985, *31*, n° 7, p. 852-869.

V. A. PETROV, Flow Line Group Production Planning, London; Business Publication Ltd., 1966.

C. PROUST, Influence des idées de S. M. Johnson sur la résolution de problème d'ordonnancement de type $n/m/F$, contraintes diverses/C_{max}, Rapport Interne, Labo d'Info. Université de Tours, 1989.