Ordonnancement d'atelier de type flow shop hybride avec tâches multiprocesseurs

ASMA LAHIMER[1], PIERRE LOPEZ[1], MOHAMED HAOUARI[2]
[1]CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse, France
Université de Toulouse ; UPS, INSA, INP, ISAE, UT1, UTM ; LAAS ; F-31077 Toulouse
asma.lahimer@laas.fr, pierre.lopez@laas.fr
[2]INSAT ; Institut National des Sciences Appliquées et de Technologie
Centre Urbain Nord BP 676 - 1080 Tunis Cedex, Tunisie
mohamed.haouari@ozyegin.edu.tr

Résumé **- Cet article concerne la résolution d'un problème d'ordonnancement d'atelier complexe de type flowshop hybride avec tâches multiprocesseurs. Le critère à optimiser est la minimisation de la durée totale d'ordonnancement (makespan). Nous proposons une méthode de recherche arborescente à base de divergences pour le résoudre. La méthode est évaluée sur des jeux-tests en mesurant son pouvoir de résolution exacte ou l'écart à une borne. Une étude comparative est également menée avec d'autres méthodes de la littérature. Les résultats obtenus prouvent l'efficacité de notre approche notamment sur les grandes instances.**

Abstract **- This paper considers multiprocessor task scheduling in a multistage hybrid flow-shop environment. The problem even in its simplest form is NP-hard in the strong sense. The great deal of interest for this problem, besides its theoretical complexity, is animated by needs of various manufacturing and computing systems. We propose a new approach based on limited discrepancy search to solve the problem. Our method is tested with reference to a proposed lower bound as well as the best known solutions in literature. Computational results show that the developed approach is efficient in particular for large problems.**

Mots-clés **- Ordonnancement, flow shop multiprocesseur, recherche à divergences.**
Keywords **- Scheduling, Hybrid Flow-Shop, Multiprocessor Tasks, Discrepancy Search.**

1 INTRODUCTION

La résolution du problème de flow shop hybride avec tâches multiprocesseurs est un problème complexe d'ordonnancement d'atelier. D'un point de vue théorique, il s'agit d'étudier une généralisation d'un problème d'atelier à cheminement unique (flow shop) et un cas particulier du problème d'ordonnancement de projet à moyens limités (RCPSP). Son étude est donc tout d'abord justifiée par une forte complexité théorique puisque le problème est NP-complet. D'un autre côté, ses champs d'application sont divers et nombreux, dans le domaine industriel et les systèmes de calcul ; un cas d'application privilégié se rencontre dans les systèmes de vision temps réel.

Le problème du flow shop hybride avec tâches multiprocesseurs est résolu dans la littérature par différentes approches (algorithme génétique [Öguz & Ercan, 2005], recherche tabou, colonies de fourmis [Şerifoğlu & Ulusoy, 2006]). Motivés par l'efficacité des méthodes de recherche arborescente à base de divergences sur les problèmes d'ateliers avec flexibilité sur les ressources, notamment les problèmes de flow shop hybride [Hmida *et al.* , 2007], nous proposons dans cette communication d'adapter ce type d'approche pour résoudre le problème plus général avec tâches multiprocesseurs.

2 DÉFINITION DU PROBLÈME

Le flow shop hybride avec tâches multiprocesseurs combine les propriétés des problèmes de type flow shop, machines parallèles et cumulatif. Formellement, le problème du flow shop hybride (FSH) avec tâches multiprocesseurs se définit comme un ensemble $\mathbf{J}=\{1,2,\ldots,n\}$ de n jobs et un ensemble $\mathbf{M}=\{1,2,\ldots,m\}$ de m étages. Le problème étudié consiste en l'ordonnancement de ces n jobs composés de m opérations (une par étage). Chaque étage $\mathbf{i}=\{1,2,\ldots,m\}$ est constitué d'un ensemble de m_i processeurs parallèles et identiques. A un étage i, le job j requiert $size_{ij}$ processeurs simultanément. Donc, $size_{ij}$ machines sont affectées à un job j au niveau de l'étage i, et assurent son traitement pendant une période de temps p_{ij}. La fonction objectif considérée est la minimisation de la durée totale d'ordonnancement (*makespan*).

La figure 1 illustre le fonctionnement d'un atelier de type flow shop hybride avec tâches multiprocesseurs. L'ensemble de départ représente les jobs à ordonnancer alors que l'ensemble d'arrivée est constitué par les produits finis.

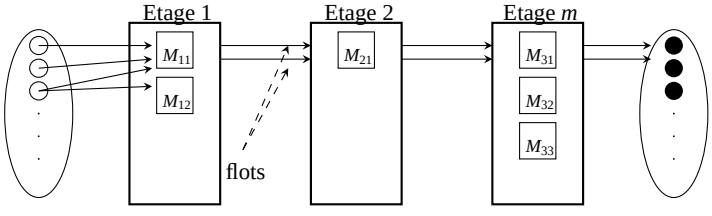

Figure 1. Flow shop hybride multiprocesseur

3 DOMAINES D'APPLICATIONS

Le problème que nous étudions est largement présent dans l'industrie, manufacturière notamment. Un fonctionnement spécifique d'une usine peut être modélisé en un flow shop hybride avec tâches multiprocesseurs, le processus de fabrication devant comporter *m* traitements consécutifs. A chaque étape est associé un ensemble de machines (un étage *i*) pouvant assurer le traitement *i*. Selon la nature de produit, la consommation de ressources (machines) diffère. Dans ce contexte, des objectifs classiques que rencontrent les entreprises sont la minimisation des encours de fabrication et la maximisation des débits de production. Pour ce dernier objectif, cela revient donc à résoudre un problème de minimisation du makespan dans un flow shop hybride avec tâches multiprocesseurs. Les industries de ce type sont nombreuses et variées, notamment le textile et la fabrication de circuits imprimés.

Une autre application privilégiée se rencontre dans les systèmes de vision temps réel. Ce type de système commence par une phase d'acquisition des images sur une scène à partir de plusieurs capteurs. Cette première étape est suivie par une modélisation des formes et des mouvements de la scène. Ensuite, les tâches d'extraction de la géométrie de cette scène sont affectées à une grappe de calculateurs. L'abstraction de ce système de vision le ramène typiquement à un problème de flow shop hybride avec tâches multiprocesseurs où les différentes phases de traitements sont modélisées par des étages, et où les calculateurs, les caméras et les capteurs jouent le rôle des ressources de l'atelier qui sont disponibles en plusieurs exemplaires. La complexité pratique d'un tel système permet de percevoir la forte complexité théorique du problème de flow shop hybride avec tâches multiprocesseurs (il est en effet NP-complet).

4 MÉTHODE PROPOSÉE

4.1 Recherche à divergences

Harvey et Ginsberg sont les initiateurs de la recherche à base de divergences en 1995. Ils proposent dans [Harvey & Ginsberg, 1995] la recherche à divergences limitées (LDS). L'objectif de LDS est de proposer le parcours d'un arbre de recherche en alternative aux procédures par séparation et évaluation (*branch-and-bound*), aux techniques de retour-arrière chronologique (*backtracking*), aux techniques d'échantillonnage itératif (*iterative sampling*). Le principe de base derrière cette technique particulière est similaire à celui d'une recherche par voisinages. A partir d'une solution initiale de référence, on réalise des écarts, ou *divergences*, pour produire une solution de meilleure qualité. Une divergence traduit ainsi une contradiction du choix de l'heuristique d'instanciation ayant permis la génération de la solution de référence. Le principe de la méthode, illustré par la figure 2, consiste lors d'une première itération à chercher tous les chemins qui divergent en une seule décision par rapport à la solution de référence. Ensuite, il s'agit de trouver tous les chemins à deux divergences de cette solution, et ainsi de suite. La première ligne en dessous des feuilles de l'arbre correspond à l'ordre de visite des nœuds et la deuxième ligne affiche le nombre de divergences effectuées pour atteindre cette feuille.

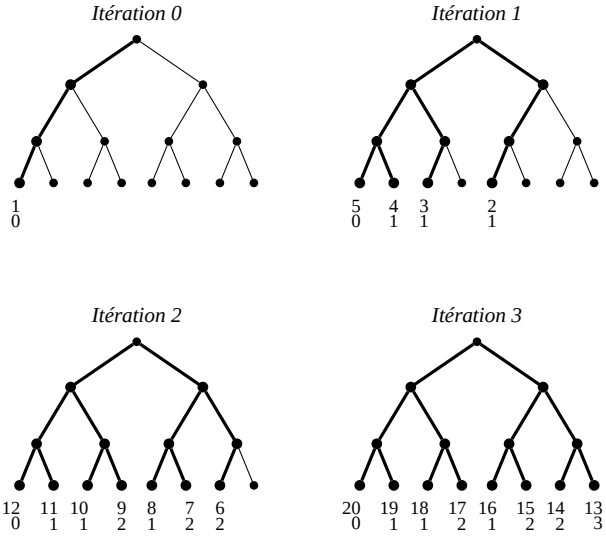

Figure 2. Limited Discrepancy Search

Depuis l'apparition de LDS, plusieurs stratégies d'exploration à base de divergences ont été introduites. Nous en examinons plus particulièrement deux dans la suite, dont nous tirons notre proposition.

4.1.1 Recherche à divergences limitées par la profondeur (DDS)

La Recherche à divergences limitées par la profondeur (*Depth-Bounded Discrepancy Search*, DDS) est proposée dans [Walsh, 1997]. Développée dans le contexte des problèmes de satisfaction de contraintes, cette méthode est considérée comme une amélioration de LDS qui privilégie les divergences à un niveau haut de l'arbre. Walsh argumente sa proposition par le postulat suivant : Les erreurs commises en bas de l'arbre n'ont pas une grande influence sur la qualité de la solution ; c'est plutôt les premières décisions prises qui ont un grand impact. De ce fait, DDS consiste à limiter la profondeur des niveaux à explorer.

4.1.2 Recherche par montée de divergences (CDS)

La recherche par montée de divergences (*Climbing Discrepancy Search*, CDS) apparue dans [Milano & Roli, 2002] est adaptée aux problèmes d'optimisation combinatoire. Cette méthode a été développée dans un contexte d'optimisation. CDS commence ainsi son exploration à partir d'une solution qui évolue dynamiquement au cours de la recherche. En effet, la méthode explore successivement les branches correspondant à un nombre de divergences croissant et dès qu'une solution améliorante est rencontrée, la solution de référence est mise à jour et le processus de recherche est réinitialisé.

4.2 Recherche par montée de divergences adjacentes limitées par la profondeur (CDADS)

Nous proposons la méthode CDADS (*Climbing Depth-bounded Adjacent Discrepancy Search*), obtenue par la combinaison d'une recherche à divergences limitées par la profondeur et d'une recherche par montée de divergences, à laquelle est adjointe la notion de divergences nécessairement adjacentes. Partant d'une solution de référence (*Sref*) obtenue par une heuristique, la méthode de recherche proposée explore progressivement son voisinage conformément au principe d'une recherche

à divergences limitées par la profondeur. On fixe ainsi une profondeur *d* dans l'arbre jusqu'à laquelle il est possible de réaliser des divergences. A une itération *i*, nous ne tolérons que *i* divergences au-dessus du niveau *d*. De plus, nous exigeons que ces divergences soient effectuées de manière adjacente, c'est-à-dire qu'il n'y a pas d'alternance entre décision divergente et décision de l'heuristique ; en d'autres termes, s'il y a plusieurs divergences dans une branche, elles doivent toutes se suivre. Cette hypothèse d'adjacence limite considérablement l'espace de recherche. Nous considérons également que l'heuristique ayant permis la génération de la solution initiale est bonne, et qu'ainsi seul l'environnement immédiat d'une divergence peut "accueillir" l'occurrence d'une divergence supplémentaire. Nous obtenons donc une méthode DDS tronquée basée sur la notion de divergences adjacentes notée DADS (*Depth-bounded Adjacent Discrepancy Search*).

Dans ce qui suit, nous illustrons la variante DDS que nous proposons par un exemple très simple (cf. figure 3) sur un arbre binaire à trois variables.

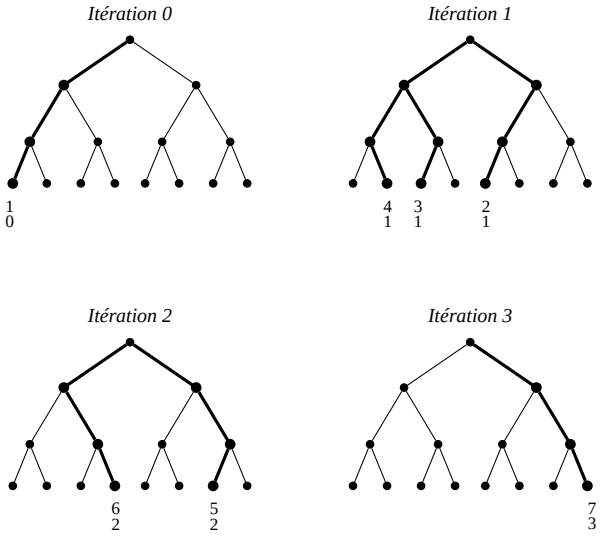

Figure 3. Trace d'exécution de DADS

L'itération 0 met en évidence la solution recommandée par l'heuristique, qui, par convention, est obtenue par la branche la plus à gauche de l'arbre de recherche. Pour une première itération, nous n'autorisons qu'une seule divergence. Les numéros affichés sur les feuilles de l'arbre expriment l'ordre de visite de chaque solution. Une deuxième itération permet de descendre dans l'arbre pour explorer d'autres branches ayant deux divergences par rapport à la solution de référence et respectant la condition d'adjacence. Sur cette illustration la profondeur autorisée est de 3. Si, par contre, nous considérons une profondeur limite de 2 niveaux, certaines branches ne seront pas retenues ; précisément, les branches 4, 6 et 7 ne seront pas visitées par la méthode DADS. Cette notion d'adjacence apparaît d'autant plus sur les arbres de taille importante.

Par exemple, soient **A**, **B**, **C** et **D** quatre jobs à ordonnancer. Nous appliquons la technique DADS pour résoudre ce problème d'ordonnancement. Nous prenons pour cela l'hypothèse qu'une divergence à un niveau *i* de l'arbre correspond à la remise en cause de la tâche occupant la position *i* dans la séquence en construction, la position relative des autres tâches de la séquence restant inchangée. L'arbre de recherche est de quatre niveaux et, pour des raisons de simplification, nous fixons la profondeur limite à 4.

- **Itération 0**. Soit la solution initiale {A,B,C,D}.
- **Itération 1**. Toutes les solutions à une seule divergence sont visitées :
  - solutions où la divergence est effectuée au premier niveau : {B,A,C,D},{C,A,B,D},{D,A,B,C} ;
  - solutions où la divergence est effectuée au deuxième niveau : {A,C,B,D},{A,D,B,C} ;
  - solutions où la divergence est effectuée au troisième niveau : {A,B,D,C}.
- **Itération 2**. Toutes les solutions à deux divergences adjacentes sont visitées : {B,C,A,D}, {B,D,A,C}, {C,B,A,D}, {C,D,A,B}, {D,B,A,C}, {D,C,A,B}.
  Les solutions {B,A,D,C}, {C,A,D,B} et {D,A,C,B} ne sont pas retenues car les deux divergences n'y respectent pas la propriété d'adjacence.

  Sur cet arbre, seules 7 solutions à deux divergences sont donc examinées, au lieu de 9 si l'on ne respecte pas la condition d'adjacence sur les divergences.
- De même pour l'itération 3 où les solutions à trois divergences adjacentes adjacentes sont considérées respectivement.

La méthode CDADS intègre le principe de la recherche DADS dans un contexte d'exploration par montée de divergences (CDS), dans la mesure où la solution de référence est mise à jour dès qu'une solution améliorante est rencontrée. DADS est donc relancée à nouveau sur cette solution, dont le voisinage est prometteur pour contenir une solution de meilleure qualité, voire une solution optimale.

4.3 Heuristiques de génération de solution de référence

L'approche CDADS proposée s'appuie grandement sur la qualité de la solution de référence. Une telle propriété justifie davantage la condition d'adjacence et conditionne la qualité de la solution finale. Le choix de l'heuristique de départ s'est donc fondé sur une étude expérimentale qui nous a permis d'élire les heuristiques les plus adaptées au problème de FSH avec tâches multiprocesseurs. Partant des études présentées dans [Şerifoğlu & Ulusoy, 2006] et [Oğuz *et al.* , 2003], les tests ont été initiés sur la base d'un ensemble de dix règles de priorité, pour en sélectionner au final les quatre plus performantes pour le problème étudié. Il s'agit de :

- SPT (*Shortest Processing Time*) : les jobs sont ordonnancés en respectant un ordre croissant de leurs durées d'exécution sur le premier étage.
- SPR (*Shortest Processing Requirement*) : les jobs sont ordonnancés en respectant un ordre croissant de leurs consommations en termes de ressources sur le premier étage.
- Énergie Croissante : l'énergie d'une opération *j* à un étage *i* étant égale à $p_{ij} \times size_{ij}$, les opérations sur le premier étage sont classées selon un ordre croissant de cette grandeur.
- NSPT (*SPT normalisée*) : cette règle de priorité est introduite dans [Şerifoğlu & Ulusoy, 2006]. Les auteurs suggèrent de classer les jobs selon un ordre croissant de leurs indices RI_j (Ranking Index) qui normalise la durée d'exécution d'un job *j* sur le dernier étage *m* par rapport à la plus grande durée d'exécution sur *m*. RI_j est calculé ainsi :

$$RI_j = \frac{\max_k p_{mk} - p_{mj} + 1}{\max_k p_{mk} + 1}.$$

Les expérimentations ont utilisé des jeux-test disponibles sur le site web de Ceyda Oğuz, `http://home.ku.edu.tr/coguz/public_html/`. Les règles retenues sont classées selon leur performance dans le tableau 1. Cette performance est mesurée en pourcentage de meilleures solutions obtenues.

Tableau 1
SÉLECTION DES HEURISTIQUES

Règle de priorité	Performance (%)
NSPT_LastStage	27
Énergie Croissante	25
SPT	17
SPR	14

4.4 Codage et décodage d'une solution

4.4.1 Codage d'une solution

Dans le cas multiprocesseur, les ordonnancements de permutation ne sont plus dominants pour le critère du makespan. Un codage logique pour une solution du Fm(m_1,...,m_m)$|size_{ij}|C_{\max}$ est la considération de m séquences de jobs où chaque séquence représente l'ordre de passage des jobs sur l'étage i ($i = 1,\ldots,m$). Cette représentation associe donc à chaque solution m séquences, chaque séquence étant une permutation de l'ensemble $\{1,\ldots,n\}$, n étant le nombre de jobs à ordonnancer.

Nous ne retenons que la séquence des jobs sur le premier étage pour coder une solution du FSH multiprocesseur. A la phase du décodage, en appliquant un algorithme de liste que nous détaillons dans le paragraphe suivant, nous obtenons l'ordre de passage des jobs sur les différents étages. Ainsi, un vecteur d'entiers de n éléments variant entre 0 et $n-1$ représente la séquence des jobs sur le premier étage de l'atelier et code une solution du problème étudié.

4.4.2 Décodage d'une solution

La phase de décodage s'appuie sur un algorithme de liste, qui, à partir d'une séquence de jobs sur le premier étage, génère un séquencement sur les m étages de l'atelier. Cet algorithme commence par attribuer les ressources du premier étage itérativement aux tâches disponibles (par hypothèse : à l'instant $t = 0$, toutes les tâches sont disponibles) en respectant les contraintes sur les ressources. Ainsi, une nouvelle liste est créée en se basant sur la date de fin des tâches sur le premier étage. Selon le nouvel ordre de passage, les tâches vont être affectées sur le deuxième étage et ainsi de suite. C'est une logique de type FCFS (First Come First Served) ou 'premier arrivé premier servi' que nous appliquons. Autrement dit, la séquence sur le premier étage ($i = 1$) est établie en respectant l'ordre recommandé par l'heuristique de départ. Ensuite, l'ordonnancement sur un étage i se base sur un ordre croissant des dates d'achèvements des différentes tâches sur l'étage $i-1$. L'affectation des tâches est aussi conditionnée par les ressources disponibles à chaque étage.

4.5 Borne inférieure

Afin de limiter l'exploration en élaguant des branches de l'arbre de recherche, nous intégrons un calcul de borne inférieure en chaque nœud. Logiquement, on a :

$$LB = \max(LB_j, LB_s).$$

LB_j est une borne basée jobs telle qu'elle se présente dans [Ŏguz & Ercan, 2005] : $LB_j = \max_{j\in J}(\sum_{i=1}^{m} p_{ij})$.
Le second terme est une borne basée étages telle que $LB_s = \max_{i=1..m} LB(i)$. Pour cette borne, et pour déterminer précisément la valeur de $LB(i)$, nous allons nous inspirer de la borne présentée dans [Ŏguz & Ercan, 2005]. Nous partons de l'hypothèse que l'ordonnancement au sein de l'atelier est assuré sans retard. Sous cette hypothèse, le temps nécessaire pour démarrer l'exécution d'un job $j \in J$ sur n'importe quelle machine de l'étage i considéré est dans le meilleur des cas égale à $\min_{j\in J}\{\sum_{l=1}^{i-1} p_{lj}\}$. De même, le temps requis pour achever l'exécution des jobs sur les étages restants (de $i+1$ à m) est au minimum $\min_{j\in J}\{\sum_{l=i+1}^{m} p_{lj}\}$. Il reste à évaluer une borne sur la durée de traitement des jobs sur l'étage i. Soit $M_1(i)$ la charge moyenne dans le cas 'préemptif'. On a : $M_1(i) = \left\lceil \frac{1}{m_i} \sum_{j\in J} p_{ij} size_{ij} \right\rceil$. On doit aussi considérer un ensemble $M_2(i)$ qui distingue les tâches traitées à l'étage selon deux ensembles de jobs A_i et B_i. Le premier ensemble regroupe les jobs nécessitant une quantité de ressources supérieure à la moitié des processeurs disponibles sur l'étage i, alors que le deuxième ensemble est constitué des jobs demandant un $size_{ij}$ exactement égal $\frac{m_i}{2}$. On a ainsi : $A_i = \{j | size_{ij} > \frac{m_i}{2}\}$, $B_i = \{j | size_{ij} = \frac{m_i}{2}\}$. Naturellement, un job de de type A_i et un autre de type B_i s'exécutent de façon séquentielle sur l'étage i ; cette propriété de séquentialité est également assurée entre tous les jobs de A_i. Ceci permet donc d'exprimer $M_2(i)$ comme : $M_2(i) = \sum_{j\in A_i} p_{ij} + \frac{1}{2}\sum_{j\in B_i} p_{ij}$. Enfin, nous ajoutons un dernier terme, $\max_{j\in J}(p_{ij})$, qui permet de maximiser la charge de l'étage i notamment quand des jobs ayant un temps d'exécution élevé sont à ordonnancer.

Globalement, on a :

$$LB(i) = \min_{j\in J}(\sum_{l=1}^{i-1} p_{lj}) + \max(M_1(i), M_2(i), \max_{j\in J}(p_{ij})) + \min_{j\in J}(\sum_{l=i+1}^{m} p_{lj}),$$

avec les définitions de M_1 et M_2 telles que décrites ci-dessus.

5 RÉSULTATS EXPÉRIMENTAUX

5.1 Environnement d'expérimentations et conditions de tests

Pour assurer une finalité de comparaison entre la méthode que nous proposons, CDADS, et celles introduites dans la littérature, nous proposons de tester les performances de CDADS sur des jeux de données présentés par Oğuz et *al.* en 2004. Ces benchmarks sont largement utilisés dans la littérature [Şerifoğlu & Ulusoy, 2004], [Jouglet *et al.* , 2009], [Oğuz *et al.* , 2004]. Les différentes instances objet du test, peuvent avoir

un nombre de jobs $n = 5, 10, 20, 50, 100$ alors que m, le nombre d'étages de l'atelier prend sa valeur dans l'ensemble $\{2, 5, 8\}$. Deux types de problèmes particuliers sont disponibles : 'Type-1' et 'Type-2'. Le nombre de processeurs dans un problème de 'Type-2' est fixé à 5 sur tous les étages (pour un étage, on a $m_i = 5, \forall i = 1, \ldots, m$). Les instances 'Type-1', ont quant à elles une configuration de machines variable avec $m_i = 1, \ldots, 5$. Pour chaque combinaison de n et m, 10 instances sont générées et ceci pour chaque type de problème. Pour les différentes instances, la durée d'exécution d'une opération (p_{ij}) et la quantité de ressources requise par un job j sur un étage i ($size_{ij}$) sont générées aléatoirement et prennent respectivement leurs valeurs dans les ensembles $\{1, \ldots, m_i\}$ et $\{1, \ldots, 100\}$. Ainsi, les batteries de tests envisagées sont menées sur un ensemble de 300 instances.

La technique CDADS est implémentée en C++. Les expérimentations sont réalisées sur un PC Intel Centrino 2 Duo 2 GHz avec 4 GB de mémoire vive. Le temps maximum alloué à la recherche est de 60 secondes. La convergence vers une solution optimale arrête l'exécution ; dans le cas contraire, on retourne la meilleure solution obtenue à épuisement du temps CPU.

Notre étude expérimentale est fondée sur une batterie de tests que nous avons menés et dont nous synthétisons ci-après les résultats.

5.2 Performance de CDADS

Nous avons testé deux stratégies liées à la priorisation du positionnement des divergences dans l'arbre de recherche. Nous envisageons ainsi une stratégie *Top First* en développant les divergences de CDADS par le haut de l'arbre et une stratégie *Bottom First* où ces divergences sont appliquées en partant du bas de l'arbre. Les expérimentations prouvent que CDADS est nettement plus efficace avec une stratégie de type Top First. Ainsi, nous ne retenons que les résultats mettant en œuvre cette stratégie.

Les deux premières colonnes du tableau 2 correspondent au type de configuration des instances (n : nombre de jobs et m : nombre d'étages). La mesure *Avg %dev* illustre la moyenne du pourcentage de déviation. Cette déviation est calculée comme suit :

- pour les instances où la solution optimale est inconnue : $\frac{C_{\max} - LB}{LB} \times 100$;
- pour les instances où la solution optimale est connue : $\frac{C_{\max} - C^*_{\max}}{C^*_{\max}} \times 100$.

On considère, pour chaque instance, la meilleure solution donnée (parmi les quatre heuristiques : SPT, SPR, Energie Croissante et SPT normalisée), la moyenne des déviations (en %) et la moyenne de temps CPU (en secondes). Pour chaque instance, nous exploitons également la propriété de symétrie entre le problème FSH avec tâches multiprocesseurs et son inverse (inversion du sens de la gamme, le dernier étage devenant le premier, le premier devenant le dernier, etc.). Ainsi, nous résolvons pour chaque instance le problème dans les deux sens et nous retenons la meilleure solution obtenue.

Ces résultats mettent en évidence le comportement de CDADS en fonction du nombre de jobs n, le nombre d'étages m et le nombre de processeurs sur un étage i (m_i).

L'augmentation de m_i, autrement dit le passage d'une configuration de machines 'Type-1' à une configuration 'Type-2' est marqué par une augmentation des déviations. Globalement, les problèmes de 'Type-1' ont une déviation moyenne de 1.66% face à une moyenne de 6.39% pour les problèmes 'Type-2' (avant-dernière ligne du tableau 2). Cela peut être interprété par l'une des hypothèses suivantes :

1. la dégradation de la borne inférieure utilisée, qui reste fortement liée à la grandeur m_i, dégrade les déviations lorsque m_i augmente ;
2. l'efficacité de la méthode CDADS est moindre sur des instances caractérisées par une grande flexibilité ($m_i = 5, \forall i = 1, \ldots, m$).

Tableau 2
PERFORMANCE DE CDADS

		Problèmes 'Type-1'		Problèmes 'Type-2'	
n	m	*Avg %dev*	*CPU (s)*	*Avg %dev*	*CPU (s)*
5	2	0	< 0.1	0	< 0.1
	5	0.21	< 0.1	0.46	< 0.1
	8	1.71	< 0.1	0.5	< 0.1
10	2	0	< 0.1	1.72	< 0.1
	5	0.66	0.4	6.44	< 0.1
	8	8.47	< 0.1	9.61	0.2
20	2	0.05	0.1	3.34	3.1
	5	2.57	1.1	7.97	1.3
	8	5.11	0.2	15	1.3
50	2	0.49	2.3	1.74	4.2
	5	0.54	5	8.2	13.5
	8	1.62	6.8	12.42	33.4
100	2	0.08	11.1	3.32	22.8
	5	1.5	13.6	10.75	40.9
	8	1.86	11	14.33	47.3
Moyenne		1.66	3.44	6.78	10.53

Selon ces résultats, quand le nombre de jobs n est fixé, le nombre d'étages m conditionne le degré de déviation : l'augmentation de m s'accompagne par une augmentation des déviations. En effet, la distance entre la solution obtenue et la borne inférieure, voire la solution optimale, grandit avec la difficulté du problème qui elle-même croît avec l'augmentation du nombre d'étages considéré.

En ce qui concerne le nombre de jobs n, nous constatons que sa variation n'a pas un effet significatif sur la déviation quand le nombre d'étages m est fixé.

Le tableau 2 illustre parfaitement les performances de notre méthode en termes de temps CPU. La limite CPU est fixée à 60 secondes et la moyenne de temps CPU varie entre 0 secondes et 47.3 secondes. La variation de temps CPU entre les problèmes 'Type-1' et les problèmes 'Type-2' confirment la complexité de ces derniers. De même, pour un nombre d'étages fixé m, quand n croît, CDADS prend plus de temps pour converger. Inversement, pour n fixé, lorsque m augmente la consommation en termes de temps CPU augmente. Exceptionnellement, les instances relatives à $n = 20$ échappent à cette règle. Ceci est dû à une valeur aberrante de 3.1 s qui correspond en réalité à une moyenne entre 9 valeurs nulles (CPU vaut 0 seconde sur 9 instances) et une valeur de 30 secondes.

5.3 *Comparaison entre CDADS et les méthodes de la littérature*

Les résultats obtenus sont comparés par rapport à ceux fournis par certaines méthodes de la littérature. Nous nous limitons aux approches proposées par [Jouglet *et al.* , 2009] qui, à notre connaissance, sont les plus récentes. La première méthode mise en œuvre dans [Jouglet *et al.* , 2009] est une implémentation d'un algorithme génétique (AG) semblable à celui de [Ŏguz & Ercan, 2005]. Une approche dite CP intègre une procédure par séparation et évaluation dans un schéma de programmation par contraintes. Enfin, les auteurs proposent un algorithme mémétique (AM) qui utilise l'algorithme génétique pour assurer la recherche de meilleures séquences et exploite une propagation de contraintes pour trouver les ordonnancements correspondants. Nous ignorons les résultats publiés par Ercan et *al.* en 2005 [Ŏguz & Ercan, 2005] compte tenu de certaines incohérences que nous avons relevées. Dans tout ce qui suit, nous confrontons nos résultats à ceux de [Jouglet *et al.* , 2009]. Cependant, nous ne retenons pas les moyennes de déviation qui y sont publiées, en raison d'erreurs et d'incohérences encore détectées (provenant des erreurs initiales faites par Ercan). Nous recalculons ainsi ces mesures sur une nouvelle base de bornes inférieures (LB) éliminant ainsi tout impact des valeurs de LB d'Ercan sur les mesures analytiques de [Jouglet *et al.* , 2009]. Le temps CPU maximum accordé dans les différents algorithmes par rapport auxquels nous menons l'étude comparative est de 900 secondes par instance.

Dans une première partie, nous comparons les différentes méthodes en termes de qualité de solutions en exploitant les mesures de déviations (voir tableau 3). Notre méthode trouve tout son intérêt en comparant ses résultats avec ceux de l'algorithme génétique. CDADS surpasse en effet AG dans 27 configurations (*n*,*m*) parmi 30. De plus, pour les configurations où AG s'avère meilleur, les performances de CDADS sont assez proches de celles de AG. Globalement, notre méthode se distingue aussi bien pour les problèmes de 'Type-1' que ceux de 'Type-2' : une déviation moyenne de 1.66% (CDADS) face à 2.27 % (AG) pour les instances de 'Type-1' et une déviation moyenne de 6.39% (CDADS) contre 8.32% (AG) pour les instances de 'Type-2'.

L'analyse des performances de CDADS par rapport à l'approche CP révèle l'efficacité de notre méthode, en particulier sur les instances de grande taille (à partir de 20 jobs et indépendamment du nombre d'étages m et de la configuration des machines m_i). L'amélioration de la déviation moyenne apparaît d'autant plus sur les instances de 'Type-2'. Ainsi, notre méthode semble adaptée à des instances qui sont à la fois de grande taille et de flexibilité importante. Pour les instances de petite taille, notamment $n = 5$ et $n = 10$, l'approche CP surpasse CDADS. Cependant, dans l'ensemble, CDADS offre des résultats meilleurs : sa déviation moyenne est strictement inférieure à la moitié de la déviation moyenne de CP (1.66% face à 5.39%) sur les problèmes de 'Type-1' et est presque égale à la moitié sur les problèmes de 'Type-2' (6.39% face à 11.92%).

La dernière phase de l'étude comparative s'intéresse à l'algorithme mémétique de [Jouglet *et al.* , 2009]. Nous confirmons, à quelques exceptions près, le comportement efficace de CDADS sur les grandes instances (au-delà de $n = 10$) et quelques insuffisances sur les petites instances (pour $n = 5$ et $n = 10$). En outre, la déviation moyenne obtenue par CDADS est de l'ordre de 1.66% alors que celle donnée par l'algorithme mémétique vaut 1.6% (pour les instances de 'Type1'). Pour les instances de 'Type2', le degré d'efficacité est inversé car CDADS offre une moyenne de déviation égale à 6.39% contre 7.28%.

En termes de qualité de solutions, nous mettons l'accent sur le fait que CDADS arrive à améliorer toutes les meilleures solutions connues (instance par instance) pour $n = 100$ (appartenant à 'Type-1' ou à 'Type-2') et $n = 50$ (uniquement pour les instances 'Type-2'). Cette constatation vient consolider notre conclusion sur l'efficacité de CDADS sur les grandes instances.

Dans un deuxième temps, les performances en termes de CPU sont examinées (voir dernière ligne du tableau 3). La nature de la méthode CDADS basée essentiellement sur une recherche arborescente limitée par le nombre de divergences la positionne en tête de toutes les méthodes étudiées. En effet, elle assure ses résultats en une moyenne de temps CPU variant entre 0 secondes, pour $n = 5$, et 47.8 secondes pour les instances les plus difficiles ($n = 100, m = 8, m_i = 5$) face à un intervalle de [0.7 sec ; 900 sec] pour l'algorithme mémétique. Les différentes méthodes objet de comparaison n'ont toutefois pas été implémentées sur le même type d'ordinateur ; le calcul du coefficient de normalisation adéquat suivant les tables de Dongarra [Dongarra, 2009] confirme néanmoins nos conclusions, puisque le rapport entre les performances théoriques des deux machines serait de l'ordre de 3.5 seulement.

6 CONCLUSION

Dans cet article, nous proposons une méthode de recherche arborescente basée sur la notion de divergences pour la résolution du problème d'ordonnancement de type flow shop avec tâches multiprocesseurs. La particularité de notre méthode est basée sur une caractéristique d'adjacence sur les divergences réalisées. Nous présentons des heuristiques pour la génération de la solution de référence, ainsi qu'une borne inférieure permettant de restreindre l'espace de recherche des solutions. Comparée à l'état de l'art, notre méthode s'avère supérieure en termes de qualité de solution obtenue et de temps de résolution.

Une perspective à court terme de ce travail est d'évaluer la méthode sur les problèmes plus simples de flow shop hybride classique ($size_{ij} = 1, \forall i, j$) pour lesquels de nombreux travaux existent. A plus long terme, une piste prometteuse concerne l'application de ce type de méthode à des problèmes plus généraux tels que le problème d'ordonnancement de projet à moyens limités (RCPSP) dont la résolution reste actuellement un des challenges majeurs en ordonnancement pour des problèmes de taille importante. Cette extension devrait sans doute passer par la mise en place de mécanismes additionnels à la méthode, tels qu'une procédure *multi-start* permettant de repartir d'une nouvelle solution de référence si une amélioration de la solution n'est pas très rapidement trouvée, ou la définition de nouvelles bornes inférieures.

Tableau 3

COMPARAISON DE LA DÉVIATION MOYENNE ET DU TEMPS CPU

		Problèmes 'Type-1'				Problèmes 'Type-2'			
n	*m*	*CDADS*	*AG*	*CP*	*AM*	*CDADS*	*AG*	*CP*	*AM*
5	2	0	0.29	0	0	0	1.23	0	0
	5	0.21	1.35	0	0	0.46	1.44	0	0
	8	1.71	4.15	0	0	0.5	2.38	0	0
10	2	0	0	0	0	1.72	2.83	1.72	1.75
	5	0.66	1.64	0	0	6.44	7.8	6.1	5.67
	8	8.47	9.38	10.32	8.02	9.61	10.87	8.37	8.8
20	2	0.05	0.44	2.59	0.66	3.34	3.7	6.72	3.43
	5	2.57	3.49	10.85	2.78	7.97	9.57	22.86	9.57
	8	5.11	5.69	17.98	5.32	15	17.26	28.52	16.02
50	2	0.49	0.63	2.79	0.49	1.74	2.76	6.54	2.21
	5	0.54	0.59	5.3	0.51	8.2	10.95	20.01	10.32
	8	1.62	2.17	14.42	1.71	12.42	15.89	30.06	17.25
100	2	0.08	0.15	1.96	0.07	3.32	3.05	5.68	2.7
	5	1.5	2.5	5.19	2.33	10.75	14.95	19.13	14.37
	8	1.86	1.99	9.47	2.15	14.33	20.06	23.15	17.83
Avg %dev		**1.66**	2.27	5.39	1.6	**6.39**	7.28	11.92	8.32
CPU (s)		**3.44**	879.93	320.3	326.01	**10.53**	879.08	423.09	511.27

RÉFÉRENCES

[Şerifoğlu & Ulusoy, 2004] Şerifoğlu, F.S., & Ulusoy, G. 2004. Multiprocessor Task Scheduling in multistage hybrid Flow-Shops : A Genetic Algorithm Approach. *European Journal of Operational Research*, **55**(5), 504–512.

[Şerifoğlu & Ulusoy, 2006] Şerifoğlu, F.S., & Ulusoy, G. 2006. Multiprocessor Task Scheduling in multistage hybrid Flow-Shops : an ant colony system approach. *International Journal of Production Research*, **44**(16), 3161–3177.

[Dongarra, 2009] Dongarra, Jack J. 2009. *Performance of Various Computers Using Standard Linear Equations Software*. Tech. rept. University of Tennessee.

[Harvey & Ginsberg, 1995] Harvey, W.D., & Ginsberg, M.L. 1995 (August). Limited Discrepancy Search. *Pages 607–615 of : Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, vol. 1.

[Hmida *et al.* , 2007] Hmida, A. Ben, Huguet, M.-J., Lopez, P., & Haouari, M. 2007. Climbing Depth-bounded Discrepancy Search for Solving Hybrid Flow Shop Scheduling Problems. *European Journal of Industrial Engineering*, **1**(2), 223–243.

[Jouglet *et al.* , 2009] Jouglet, A., Ŏguz, C., & Sevaux, M. 2009. Hybrid Flow-Shop: a Memetic Algorithm Using Constraint-Based Scheduling for Efficient Search. *Journal of Mathematical Modelling and Algorithms*, **8**, 271–292.

[Milano & Roli, 2002] Milano, M., & Roli, A. 2002. On the relation between complete and incomplete search : an informal discussion. *Page 237–250 of : Proceedings of the 4th International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'02)*.

[Oğuz *et al.* , 2003] Oğuz, C., Ercan, M.F., Cheng, T.C.E., & Fung, Y.F. 2003. Heuristic algorithms for multiprocessor task scheduling in a two stage hybrid flow shop. *European Journal of Operational Research*, **149**, 390–403.

[Oğuz *et al.* , 2004] Oğuz, C., Y.Zinder, Do, V. Ha, Janiak, A., & Lichtenstein, M. 2004. Hybrid flow shop scheduling problems with multiprocessor task systems. *European Journal of Operational Research*, **152**, 115–133.

[Ŏguz & Ercan, 2005] Ŏguz, C., & Ercan, M.F. 2005. A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks. *Journal of Scheduling*, **8**, 323–351.

[Walsh, 1997] Walsh, T. 1997 (August). Depth-bounded Discrepancy Search. *Page 1388–1395 of : Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, vol. 2.