Recherches arborescente et locale pour les problèmes d'ordonnancement avec contraintes de précédence et temps de préparation

Bernat Gacias, Christian Artigues, Pierre Lopez

Recherches arborescente et locale pour les problèmes d'ordonnancement avec contraintes de précédence et temps de préparation

B. Gacias, C. Artigues et P. Lopez

LAAS-CNRS, Université de Toulouse, 31077 Toulouse CEDEX 4
{bgacias,artigues,lopez}@laas.fr

Résumé Cet article concerne l'étude de différentes méthodes de résolution pour les problèmes d'ordonnancement d'opérations sur machines parallèles avec contraintes de précédence et temps de préparation des machines entre l'exécution des différentes opérations. Des méthodes de recherche arborescente à divergences limitées intégrant des concepts de recherche locale, des conditions de dominance et des bornes inférieures spécifiques sont proposées et validées sur des jeux de données générés aléatoirement.

Mots-Clefs. Ordonnancement à machines parallèles, temps de préparation, recherche à divergences limitées, recherche locale.

1 Introduction

Le problème étudié dans cet article est l'ordonnancement d'opérations sur machines parallèles avec contraintes de précédence et temps de préparation des machines entre l'exécution des différentes opérations. Nous considérons l'optimisation de deux critères différents : la minimisation de la somme des dates de fin de chaque tâche et la minimisation du retard algébrique maximal sur l'ensemble des tâches. Ces fonctions objectif présentent un intérêt particulier en ordonnancement de la production. La somme des dates de fin est un objectif qui maximise le débit de production et qui permet de minimiser les stocks des produits intermédiaires ou d'en-cours, produits non terminés, présents dans l'atelier. Dans la minimisation du retard algébrique maximal, les dates d'échéance peuvent correspondre aux dates de livraison des produits. C'est donc un objectif de respect des délais qui vise à pénaliser le moins possible le client livré le plus en retard. Ces problèmes sont *NP-difficiles* au sens fort [5].

Le problème à machines parallèles a été largement étudié [3], notamment parce qu'il apparaît comme une relaxation de problèmes plus complexes comme le flow shop hybride ou le RCPSP (Resource-Constrained Projet Scheduling Problem). Dans la littérature, plusieurs méthodes ont été proposées pour la résolution de ce problème. Chen et Powell [2] résolvent le problème par une méthode de génération de colonnes. Salem *et al.* [15] proposent des méthodes basées sur la recherche arborescente. Dans ce même contexte et plus récemment, Tercinet *et al.* [14] comparent deux schémas de branchement différents et aussi plusieurs méthodes de recherche locale.

Le problème avec contraintes de précédence et temps de préparation a été beaucoup moins étudié. La présence simultanée des deux types de contraintes augmente de façon sensible la complexité du problème. Les problèmes intégrant une seule de ces deux limitations peuvent être résolus par un algorithme de liste, c'est-à-dire qu'il existe une combinaison des tâches, qui, lorsqu'on suit une certaine règle d'affectation et d'ordonnancement, aboutit à la solution optimale [16]. La règle à

suivre est Earliest Completion Time (ECT), qui consiste à affecter la tâche à la machine qui permet de la finir au plus tôt en plaçant la tâche après toutes les tâches déjà ordonnancées sur cette machine. Ce raisonnement n'est plus valide lorsque des contraintes de précédence et de temps de préparation entre les tâches sont considérées comme cela est démontré en [8], ce qui oblige à modifier la façon de traiter le problème.

Dans le paragraphe 2, le problème à machines parallèles avec temps de préparation et contraintes de précédence entre les opérations est défini formellement. Les méthodes et techniques de recherche locale et arborescente appliquées pour la résolution du problème sont décrites dans les paragraphes 3 et 4. Le paragraphe 5 est dédié aux tests réalisés.

2 Position du problème

Il s'agit de réaliser l'affectation de n tâches sur l'ensemble de m machines disponibles. Les relations de précédence entre les différentes tâches doivent être respectées, ainsi que les temps de préparation rencontrés lorsque les tâches sont affectées de façon consécutive sur la même machine. Chaque machine exécute au maximum une tâche simultanément et chaque tâche ne peut pas être exécutée par plus d'une machine à un instant de temps donné. Dans notre problème, la préemption n'est pas autorisée, c'est-à-dire que chaque tâche i s'exécute durant p_i unités de temps consécutives sur la même machine. Les variables de décision du problème sont S_i et C_i, date de début et date de fin d'exécution de la tâche i respectivement.

Les données du problème sont :

- J : ensemble $\{1, \ldots, n\}$ de tâches ;
- p_i : durée de la tâche i ;
- r_i : date de début au plus tôt de la tâche i ;
- d_i : date de fin au plus tard de la tâche i ;
- M : ensemble $\{1, \ldots, m\}$ de machines ;
- E : ensemble modélisant les contraintes de précédence entre les tâches. La relation $(i, j) \in E$, avec i et $j \in J$, signifie que la tâche i précède la tâche j (ce qui se note formellement $i \prec j$). La tâche j peut ainsi débuter au plus tôt après la fin de la tâche i ($S_j \geq C_i$) ;
- s_{ij} : temps de préparation de la tâche j effectuée immédiatement après la tâche i sur une machine. Pour deux tâches i et j réalisées de façon consécutive sur la même machine, alors $S_j \geq C_i + s_{ij}$ si i précède j ou $S_i \geq C_j + s_{ji}$ si j précède i.

Compte tenu de ces notations, les problèmes étudiés s'expriment : $P|prec, s_{ij}|\sum C_i$ et $P|prec, s_{ij}|L_{\max}$.

Exemple

Un ensemble de 5 tâches ($n = 5$) doit être réalisé sur 2 machines différentes ($m = 2$). Pour chaque tâche i, on donne dans le tableau 1 la valeur des différentes grandeurs p_i, r_i, d_i et s_{ij}. De plus, pour cet exemple, on a les contraintes de précédence suivantes : $1 \prec 4$ et $2 \prec 5$.

(a)

n	p_i	r_i	d_i
1	4	1	7
2	3	0	5
3	4	2	8
4	3	3	10
5	2	1	5

(b)

s_{ij}	**1**	**2**	**3**	**4**	**5**
1	0	2	3	4	2
2	3	0	6	1	3
3	1	2	0	3	4
4	3	4	10	0	5
5	3	4	10	5	0

Tab. 1. Données de l'exemple illustratif

La figure 1 représente une des solutions possibles pour ce problème.

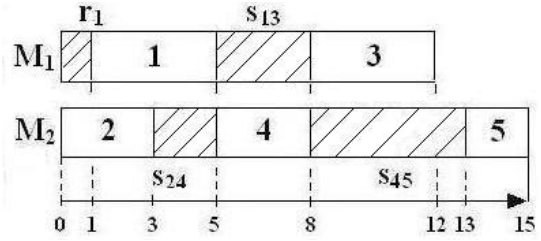


Fig. 1. Exemple d'un ordonnancement réalisable

Dans cette solution, l'ensemble de contraintes de précédence est respecté : $S_5 = 13 \geq 3 = C_2$ et $S_4 = 5 \geq 5 = C_1$. Remarquons que la tâche 4 se voit forcée à retarder d'une unité son début à cause d'une contrainte de précédence. D'un autre côté, il faut vérifier que, pour chaque tâche, $r_i \leq S_i$ et que les temps de préparation entre tâches consécutives sur une même machine sont aussi respectés. En ce qui concerne la fonction objectif, notre critère d'évaluation de la qualité de la solution, observons que pour la minimisation de la somme des dates de fin des tâches la fonction vaut $z = \sum C_i = 43$ et pour le cas de la minimisation du retard algébrique maximal, $z = L_{\max} = L_5 = 10$.

3 Recherche basée sur les divergences et recherche locale

Les principes des méthodes basées sur les divergences et de la recherche locale sont présentées dans les sections suivantes. Nous présentons aussi les différentes méthodes d'hybridation de ces deux méthodes proposées pour la résolution du problème.

3.1 Les méthodes de recherche arborescente basées sur les divergences

La résolution du problème présenté dans le paragraphe précédent peut être réalisée par des méthodes de recherche arborescente basées sur les divergences par rapport à une heuristique préalable. Ces méthodes sont basées sur la bonne performance de l'heuristique choisie. Puis une recherche locale autour de la solution donnée par cette heuristique est réalisée de manière ordonnée en examinant d'abord les solutions avec le moins de divergences par rapport à cette solution heuristique, puis en s'éloignant jusqu'à couvrir tout l'espace de recherche autorisé.

La méthode *LDS* ("Limited Discrepancy Search" ou Recherche à Divergences Limitées) [6] a été conçue initialement comme une méthode itérative pour la résolution des problèmes à variables

binaires par recherche arborescente, où chaque nœud de l'arbre a deux nœuds descendants représentant respectivement les décisions d'affecter une variable aux valeurs 0 (non) ou 1 (oui). L'autre principe de base de cette méthode est d'explorer d'abord les solutions qui présentent les divergences dans les niveaux les plus hauts de l'arbre. En effet, les erreurs les plus importantes de l'heuristique sont celles réalisées dans les premiers niveaux où peu de décisions ont encore été prises.

A partir du principe général de LDS, plusieurs instanciations ont été proposées. La méthode *ILDS* [10] cherche à éviter la redondance en ne développant pas la recherche à partir de tout nœud menant obligatoirement à des solutions ayant un nombre de divergences strictement inférieur au maximum autorisé.

Le principe de la méthode *DDS* [17] propose un autre moyen de limiter la redondance renforçant le principe consistant à favoriser en premier les divergences situées en haut de l'arbre de recherche ; on autorise des divergences uniquement pour les niveaux de profondeur inférieure à l'itération courante. Finalement, on trouve la méthode *DBDFS* [1] qui permet elle aussi de réduire les redondances lors des itérations et la méthode *YIELDS* [9] où des notions d'apprentissage sont intégrées.

3.2 Les méthodes de recherche locale à grands voisinages basées sur LDS

Dans une méthode de recherche locale, on définit un voisinage $N_k(x)$ d'une solution x déjà trouvée (k définit les variations admissibles par rapport à la solution x). Si l'on trouve une solution x' meilleure que x dans $N_k(x)$ alors on traite le voisinage $N_k(x')$ de cette nouvelle solution.

Pour les problèmes de grande taille, les voisinages à explorer deviennent trop vastes, et on peut considérer la recherche de la meilleure solution dans le voisinage $N_k(x)$ (où k est de grande taille) comme un sous-problème d'optimisation. Dans ce cas, on peut envisager une méthode basée sur les divergences pour résoudre ce sous-problème. Les deux principales difficultés à prendre en compte lorsqu'on définit une recherche à grands voisinages sont, dans un premier temps, de trouver un compromis entre le temps de calcul et la qualité des solutions auxquelles on arrive. Il nous faut alors choisir des algorithmes qui améliorent la possibilité de trouver une meilleure solution dans le voisinage exploré. Or, dans les problèmes de grande taille, arriver à trouver l'optimum global est très difficile.

La philosophie de LDS peut être vue comme un principe de recherche locale autour de la solution définie par l'heuristique. C'est l'idée qu'exploite la méthode *CDS (Climbing Discrepancy Search)* [12] (voir algorithme 1), basée sur le même principe que la recherche locale, mais qui utilise les divergences par rapport à la meilleure solution courante pour la réalisation de la recherche. Nous trouvons dans la littérature d'autres méthodes, notamment *CDDS (Climbing Depth-bounded Discrepancy Search)* [7] qui s'est avérée efficace sur des problèmes de Flow-Shop hybride. *CDDS* intègre les principes de *CDS* et de *DDS*. Le voisinage de la meilleure solution actuelle est limité non seulement par le nombre de divergences mais aussi par la profondeur de l'arbre. De cette façon, on bénéficie du fait que les erreurs les plus graves commises par l'heuristique se trouvent en haut de l'arbre pour limiter et améliorer l'espace de recherche.

begin
$k \leftarrow 0$;
$k_{max} \leftarrow n$;
$Sol_{ref} \leftarrow HeuristiqueInitiale()$;
while $k \leq k_{max}$ **do**
$k \leftarrow k + 1$;
Générer l'ensemble de solutions N à k divergences de Sol_{ref}
$N = LDS(Sol_{ref}, k)$;
$s' \leftarrow MeilleurDe(N)$;
if $z(s') < z(Sol_{ref})$ **then**
$Sol_{ref} \leftarrow s'$;
$k \leftarrow 0$;
end

Algorithm 1: Algorithme *CDS*

Dans cet article, nous proposons de résoudre le problème à partir de deux variantes de *CDS* et *CDDS* adaptées aux caractéristiques et singularités de notre problème. La recherche idéale serait une recherche qui permettrait autant de divergences que possible dans les niveaux les plus hauts de l'arbre, mais qui, en même temps, prendrait en compte le fait que le problème présente des contraintes de précédence. Dans ce cas, l'autorisation d'un nombre de divergences limité dans les niveaux plus bas de l'arbre pourrait être efficace. Les méthodes pour l'implantation de cette recherche sont les suivantes :

- *HD-CDDS* (*Hybrid Discrepancy CDDS*)
  Il s'agit de réaliser une recherche comme dans *CDDS* mais, lorsque pour un niveau maximal *dmax* on n'a pas trouvé de meilleure solution en autorisant un nombre de divergences très élevé, alors on autorise pour tous les niveaux un nombre petit de divergences.
- *MC-CDS*(*Mix Counting CDS*)
  La deuxième méthode proposée est l'application de *CDS* mais avec une modification au niveau du comptage des divergences. La proposition est de considérer un comptage binaire pour les divergences des niveaux situés au plus haut de l'arbre et un comptage non-binaire pour le reste des niveaux. De cette façon, les branches des niveaux plus hauts sont beaucoup plus explorées et, d'un autre côté, nous autorisons des divergences dans tous les niveaux mais évitons ainsi que la recherche explose.

Dans le paragraphe 4, nous décrivons les différents modes de comptage des divergences et les autres éléments de la méthode *LDS* utilisée à chaque itération de la méthode de recherche locale.

4 Schéma de branchement et évaluation des nœuds

Dans ce paragraphe, nous spécifions les différents éléments de la méthode LDS pour l'ordonnancement à machines parallèles avec contraintes de précédence et temps de préparation. La structure de l'arbre de recherche intégrant deux types de décisions (ordonnancement et allocation de ressources) est présentée en 4.1. La stratégie d'exploration de cet arbre en termes de règles de branchement, de définition et de comptage des divergences est donnée en 4.2. Des méthodes spécifiques d'évaluation des nœuds comprenant des calculs de bornes inférieures, des techniques de propagation des contraintes et des règles de dominance sont introduites en 4.3.

4.1 Définition de l'arbre de recherche

Les problèmes d'ordonnancement qui présentent conjointement des contraintes de précédence et des temps de préparation ne peuvent pas toujours être résolus efficacement par un algorithme de liste. Un algorithme de liste permet de représenter un ordonnancement uniquement sous la forme d'une liste de tâches sachant qu'il existe une règle dominante d'affectation des ressources aux tâches et d'ordonnancement des tâches dans l'ordre de la liste. La règle est dominante au sens qu'il existe une liste de tâches telle qu'une solution optimale est obtenue en suivant la règle. S'il n'y a aucune contrainte de précédence dans le problème, alors la règle consistant à prendre chaque tâche dans l'ordre de la liste et à la positionner à la fin de la séquence partielle de la machine qui permet de la terminer au plus tôt est dominante. Ainsi, il suffit de trouver la meilleure liste (parmi les $n!$ permutations). Ce n'est plus le cas avec les contraintes de précédence comme le montre le contre-exemple présenté en [8].

Dans notre cas, en plus de trouver la meilleure liste de tâches, il faut aussi déterminer la meilleure allocation de ressources. Globalement, la meilleure solution peut être trouvée par une recherche arborescente à deux niveaux de décision à partir d'un nœud représentant une solution partielle intégrant p tâches : un ensemble d'au plus $n-p$ nœuds fils correspondant à la prochaine tâche à ordonnancer parmi les tâches non ordonnancées (ce qui correspond à la construction de la liste), puis pour chacun des nœuds ainsi générés, un ensemble de nœuds fils énumérant les allocations possibles (m nœuds). La quantité $n-p$ représente une limite supérieure puisque seules les tâches dont les prédécesseurs ont été exécutés peuvent être ordonnancées. A tout moment, la liste construite respecte les contraintes de précédence entre les tâches.

En pratique, pour homogénéiser le concept de nœuds, on intègre les deux types de décision (voir figure 2) : un branchement correspond à la sélection de la prochaine tâche à ordonnancer et au choix de la ressource allouée. Un nœud de l'arbre correspond à une liste de p tâches et une allocation partielle de ces p tâches, et comporte au plus $m(n-p)$ nœuds fils. Une solution est atteinte lorsque, pour un nœud, $p = n$.

La proposition suivante permet de diminuer le nombre de nœuds de l'arbre : à un nœud de l'arbre, pour toute tâche x, nous envisageons les affectations sur les $\min(m, t+1)$ premières machines qui permettent de finir la tâche au plus tôt, où t est le nombre de successeurs (directs où indirects) de la tâche x.

Justification de la proposition
$\diamond$ Pour une tâche x sans successeurs($t = 0$), nous l'affectons sur la machine telle que l'opération finisse au plus tôt, puisque cette affectation respecte l'algorithme de liste qui fournit une solution dominante pour un problème qui ne présente que des temps de préparation (sans contraintes de précédence). Pour une opération x avec successeurs ($t > 0$), nous devons analyser dans quel cas les contraintes de précédence empêchent de créer la liste qui permet de parvenir à la solution optimale et déterminer le nombre de machines auxquelles nous devons envisager d'affecter l'opération.

Soit l'ensemble d'opérations Y ($y \in Y$) qui a comme prédécesseur l'opération x ($x \prec y, \forall y \in Y$). Soient S_{ij} et C_{ij} les dates de début et de fin d'une opération i lorsqu'elle est affectée à la machine j. Alors :

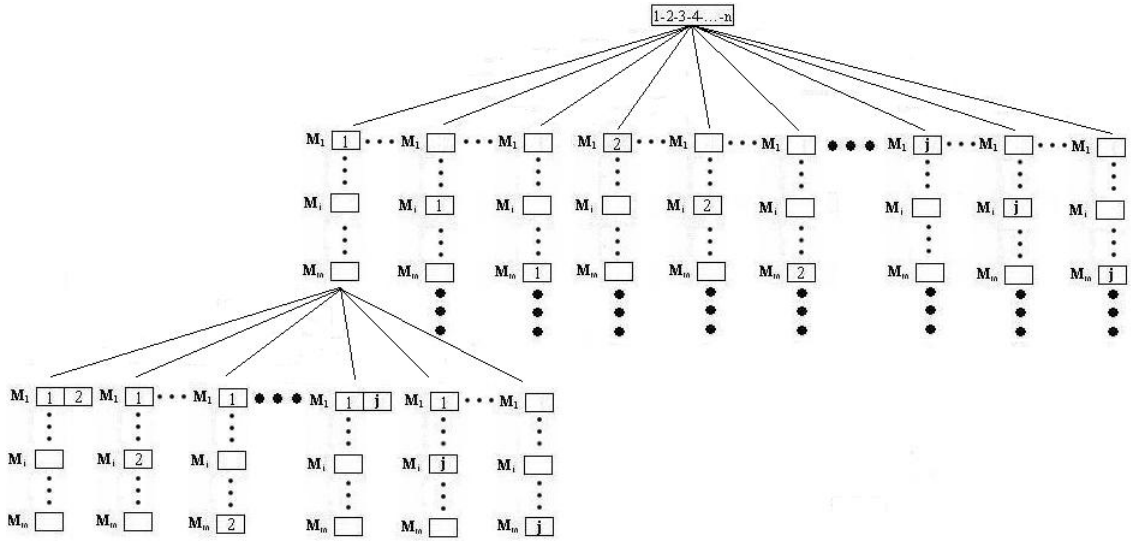


Fig. 2. Arbre de recherche développé

Affecter l'opération x selon le critère ECT sur la machine M_K ($C_{xK} < C_{xj}, \forall j = 1, \ldots, m,\ j \neq K$) est le choix optimal sauf dans le cas où M_K est aussi la machine qui permet de finir au plus tôt l'opération y ($C_{yK} < C_{yj}, \forall j = 1, \ldots, m,\ j \neq K$). Pour ce dernier cas, il est nécessaire de déterminer les deuxièmes machines qui permettent de finir les tâches au plus tôt. Si pour la deuxième machine qui permet de finir l'opération y au plus tôt (M_L), la date de début de cette opération y est supérieure à la date de fin de l'opération x sur la deuxième machine qui permet de la finir au plus tôt (M_R) (voir figure 3(a)). C'est-à-dire :

$$S_{yL} = C_{cL} + s_{cy} > C_{bR} + s_{bx} + p_x = C_{xR} \tag{1}$$

alors y doit être affectée sur la machine M_K (la machine qui permet de la finir au plus tôt). Si c'est nécessaire, il faut aussi retarder le début de l'opération y après la fin de l'opération x, afin de respecter la contrainte de précédence (voir figure 3(b)).

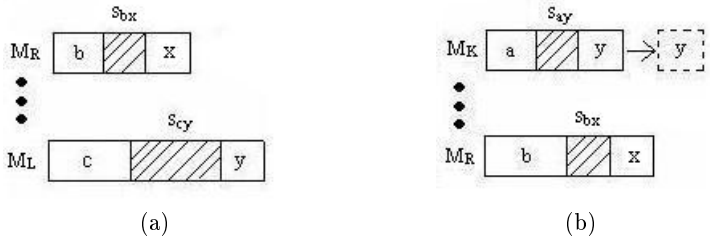


Fig. 3. Justification de la proposition

Dans l'autre cas ($C_{xR} > S_{yL}$), le meilleur choix est d'affecter x sur M_k et y sur M_L. Ainsi, lorsqu'il existe une opération x qui précède d'autres opérations, il faut l'affecter sur la machine qui permet de la finir au plus tôt, mais aussi envisager la possibilité de l'affecter sur autant de machines qu'elle a de successeurs (nombre d'éléments de l'ensemble Y). Ceci permet, pour chaque successeur $y \in Y$,

de l'affecter sur la machine qui permet de la finir le plus tôt dans les cas où cette machine est la même que pour l'opération précédente x.$\diamond$

4.2 Stratégie d'exploration

Pour la mise en œuvre, nous gérons les nœuds à explorer dans une pile. De cette façon, nous évitons totalement la redondance dans les nœuds explorés. Cela procure également une plus grande souplesse et une plus grande efficacité dans le traitement des nœuds en mémoire. D'un autre côté, la mémoire utilisée par cet algorithme est plus importante que celle requise par des techniques itératives.

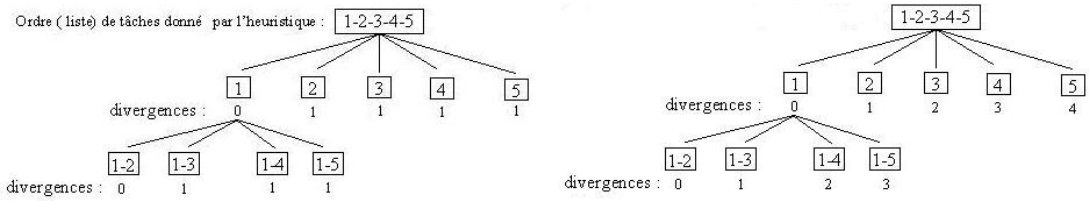
Les heuristiques d'initialisation

La solution initiale est obtenue à partir des heuristiques suivantes. Dans le cas de $\min \sum C_i$, nous utilisons, pour l'ensemble d'opérations disponibles ($r_i \leq \max_j\{tf_j\}, \forall j = 1, \ldots, m$, où tf_j est la date de fin de la dernière opération affectée à la machine j), le critère *SPT (Shortest Processing Time)* pour la sélection de l'opération et *EDD (Earliest Due Date)* dans le cas de $\min L_{\max}$. Une fois l'opération choisie, la règle à suivre pour l'affectation de l'opération aux machines est *ECT (Earliest Completion Time)*. Dans le cas où plusieurs machines permettent de finir l'opération en même temps, l'heuristique affecte l'opération à la machine de plus petit indice.

Les divergences

Il faut considérer deux types différents de divergences : les divergences sur la sélection des opérations à exécuter et les divergences sur les allocations des machines.

Dans le cas des arbres p-aires, on dispose de deux modes différents pour compter les divergences (voir la figure 4). Dans le premier mode, on considère que choisir la décision de l'heuristique correspond à 0 divergences et dans tous les autres cas la divergence est 1 ; on l'appellera mode binaire. L'autre mode de comptage, qu'on appellera mode non-binaire, consiste à considérer que plus on descend dans la liste qui représente les priorités (de sélection d'opérations ou de machines), plus on augmente le nombre de divergences.



(a) Mode binaire

(b) Mode non-binaire

Fig. 4. Illustration des deux modes de comptage des divergences pour la sélection des opérations

Nous proposons l'implémentation des deux modes de comptage pour l'heuristique de sélection des opérations. En effet, les heuristiques proposées peuvent présenter des erreurs importantes, car les temps de préparation entre opérations jouent un rôle déterminant dans l'ordonnancement des opérations et ne sont pas pris en compte dans ces heuristiques. En revanche, pour l'heuristique d'allocation, nous ne conservons que le mode non-binaire pour le comptage des divergences, puisque nous avons vu que l'heuristique ne se trompe que dans des conditions très particulières.

Stratégies de branchement

La stratégie de branchement détermine l'ordre d'exploration des nœuds lors de la recherche.

Recherche LDS en profondeur

Cette stratégie ne nécessite pas d'algorithme de recherche du prochain nœud à explorer. Elle est basée sur l'idée qu'une fois choisi le nœud fils on va réaliser l'exploration de toutes les solutions à partir de cette branche avant de passer à l'exploration des nœuds frères. Le critère pour choisir le nœud à explorer est de commencer par le nœud avec le moins de divergences par rapport à la solution initiale. Par ailleurs, cette stratégie ne respecte pas l'ordre des divergences croissantes pour les solutions explorées et ne prend pas en compte le fait que les fautes les plus graves se produisent dans les niveaux les plus hauts de l'arbre. On appellera cette stratégie *LDS-prof*.

Recherche à partir des divergences

Nous proposons deux sous-stratégies différentes d'exploration. La première, que nous appellerons *LDS-haut*, est la stratégie originale de la méthode LDS, où le nœud suivant à explorer lorsqu'on arrive à une solution est le nœud avec le moins de divergences et qui se trouve le plus haut dans l'arbre. De cette façon, si les erreurs importantes de l'heuristique se trouvent dans la partie la plus haute on va trouver les meilleures solutions plus tôt. Par contre, cette stratégie présente l'inconvénient qu'avant d'arriver à une solution elle génère et empile une grande quantité de nœuds, fait qui peut poser des problèmes de mémoire.

La deuxième sous-stratégie, *LDS-bas*, explore, après l'obtention d'une solution, le nœud qui a le moins de divergences et qui se trouve le plus bas dans l'arbre.

4.3 Evaluation d'un nœud

Le traitement de chaque nœud (ordonnancement et affectation partielle) consiste à calculer une borne inférieure pour le problème de la minimisation de la somme des dates de fin et à appliquer des techniques de coupe par raisonnement énergétique pour celui de la minimisation du retard algébrique maximal. Nous avons également proposé des règles de dominance adaptées à la recherche locale.

Borne inférieure

Pour le calcul d'une borne inférieure dans le cas de $\min \sum C_i$, nous avons choisi la borne présentée en [13] basée sur la résolution d'une relaxation du problème considéré.

Raisonnement énergétique

Pour le raisonnement énergétique [11], que nous désignerons par NRJ, l'énergie est produite par les ressources disponibles et consommée par les opérations. Dans notre problème, nous l'appliquons comme technique de coupe du nœud traité dans le cas où les solutions issues de ce nœud n'arrivent pas à améliorer la meilleure solution courante.

Il faut déterminer des fenêtres de temps pour les opérations qu'il reste à affecter. Ces fenêtres $[r_i, d'_i]$ sont fixées par la propagation des différentes contraintes à mesure que l'affectation avance.

L'énergie maximale disponible sur un intervalle de temps $\Delta = [t_1, t_2]$ correspond au produit de la longueur de l'intervalle par la quantité de ressources :

$$E_{dispo} = m \times (t_2 - t_1) \tag{2}$$

Pour la détermination de l'énergie requise, nous devons calculer pour chaque opération quelle est sa consommation obligatoire, c'est-à-dire quelle est la consommation minimale sur l'intervalle Δ de l'opération exécutée dans sa fenêtre de temps. L'expression de l'énergie requise est alors :

$$E_{requise} = \sum_i E_i = \sum_i \max(0, \min(p_i, t_2 - t_1, r_i + p_i - t_1, t_2 - d'_i + p_i)) \tag{3}$$

Afin d'intégrer dans ce terme la présence des temps de préparation, nous allons considérer une consommation qu'on appellera C_{setup}. Après une analyse du problème, nous nous sommes aperçu que, pour un intervalle de temps Δ où il y a un ensemble F de k opérations qui consomment, la quantité minimale de temps de préparation qui va apparaître est $k - m$. Ainsi, pour le calcul de la consommation des temps de préparation, nous prendrons les $k - m$ temps de préparation plus petits de l'ensemble $s_{ij}, i, j \in F$.

La condition à vérifier pour chaque ordonnancement partiel est que la somme de l'énergie requise et de la consommation due aux temps de préparation ne doit jamais être supérieure à l'énergie disponible :

$$E_{requise} + C_{setup} \leq E_{dispo} \tag{4}$$

Dans le cas contraire, le nœud sera coupé.

Le calcul de la date limite d'_i est réalisé à partir de la meilleure solution courante z_{Best}. Il s'agit de déterminer une date limite pour chaque tâche ; si la date de fin d'exécution de la tâche est supérieure à cette date limite alors la meilleure solution accessible depuis ce nœud sera toujours moins bonne que la meilleure solution courante ($z(node) > z_{Best}$) et la branche pourra être coupée.

On a :

$$L_{max} < z_{Best} \tag{5}$$

soit,

$$L_{max} = C_i - d_i < z_{Best} \tag{6}$$

et on a ainsi la date limite,

$$d'_i = C_i = z_{Best} + d_i - 1 \tag{7}$$

Pour la détermination de l'intervalle Δ, nous avons choisi de prendre $t_1 = \min\{r_i\}$ et $t_2 = \max\{d_i\}, i \in F$.

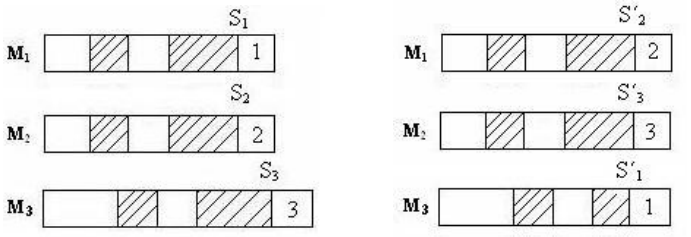
Les règles de dominance

Deux règles de dominance basées sur les concepts d'ordonnancements actifs et semi-actifs ont été envisagées pour couper l'exploration de certains nœuds. Ces règles sont adaptées à la recherche locale. Nous cherchons pour chaque nœud s'il existe une affectation partielle dominante parmi les affectations que nous allons explorer, en prenant en compte que nous sommes limité pour le nombre de divergences autorisées.

La dominance de front

Pour cette première règle (DF), le *Front* est défini comme l'ensemble des dernières tâches affectées sur chaque machine. Cette règle consiste à tester dans les nœuds dont les dernières m tâches affectées font partie du front s'il existe une combinaison de ces tâches, en respectant les contraintes de précédence et de divergences, qui permet à au moins une de ces m tâches de débuter plus tôt ($S'_i < S_i$) et de ne pas retarder la date de début initiale de toute autre tâche de l'affectation partielle,

$(S'_j \leq S_j, \forall j \neq i)$ (voir figure 5). Dans ce cas, la branche pourra être coupée. Il faut remarquer que pour couper ce nœud, il est nécessaire de maintenir le même front original. Cette règle est de complexité algorithmique $O(m!)$.



(a) affectation partielle courante

(b) affectation qui amène à une solution meilleure

Fig. 5. Le nœud courant est coupé : $S'_3 < S_3$ et $S'_j \leq S_j, j = 1, 2$

Dominance locale pour l'affectation partielle

Cette règle (*DL*) est assez similaire à la règle de *dominance de front* mais, dans ce cas, nous essayons de trouver une combinaison de la liste des opérations qui font partie de l'affectation partielle $\sigma(p)$ mais qui ne font pas partie du front (en respectant les contraintes de précédence et les divergences maximales acceptées) et qui permet de trouver une affectation $\sigma'(p)$ qui respecte les conditions précédentes : $\exists i$ $(S'_i < S_i)$ et $(S'_j \leq S_j, \forall j \in Front, j \neq i)$. Dans ce cas, les listes possibles peuvent être de taille beaucoup plus grande que pour la *dominance de front*. Le critère de modification que nous proposons est alors d'interchanger les positions des opérations qui entraînent les plus grands temps de préparation.

5 Résultats expérimentaux

Les méthodes et techniques présentées dans ce rapport ont été testées sur des instances générées aléatoirement en raison du manque d'exemples d'instances réelles pour ce type de problème dans la littérature.

Nous avons pris tout d'abord les contraintes de précédence générées par le logiciel RanGen [4] qui crée des instances différentes avec la possibilité de faire varier la densité du graphe de précédences (*OS*), afin d'avoir plusieurs niveaux de difficulté. Ce logiciel est conçu pour la génération d'instances pour le problème *RCPSP* qui ne prend en compte ni les temps de préparation entre les tâches, ni la présence de fenêtres de temps.

Ceci nous a incité à développer notre propre générateur des temps de préparation et des fenêtres de temps. Les temps de préparation doivent respecter l'inégalité triangulaire faible, soit :

$$s_{ij} \leq s_{ik} + p_k + s_{kj}, \forall i, j, k \tag{8}$$

Par ailleurs, en ce qui concerne les fenêtres de temps nous avons utilisé un schéma classiquement rencontré dans la littérature. Ainsi, d_i est généré selon une distribution uniforme $[\max(0, P \times (1 - \tau - \rho/2)), P \times (1 - \tau + \rho/2)]$, avec $P = \sum(p_i + \min_j(s_{ij}))$, $\tau \in [0, 1]$, $\rho \in [0, 1]$; les r_i sont générées à partir de ces dates limites et d'un paramètre α qui permet d'obtenir des fenêtres plus ou moins larges afin de modifier la difficulté du problème : $r_i = d_i - (p_i \times (2 + \alpha))$ où $\alpha \in [-0.5, +1.5]$. Nous proposons sis familles d'instances différentes pour les différents tests à réaliser.

Famille	n	m
1	10	3
2	15	2
3	40	2
4	40	4
5	100	2
6	100	4

Tab. 2. Les différentes familles d'instances

Pour chaque famille d'instances, nous générons 30 problèmes avec des valeurs différentes des paramètres OS, $\bar{s_{ij}}/\bar{p_i}$, α, ρ et τ afin d'avoir une représentation de toutes les classes de problèmes. Le temps de calcul est limité et, au bout du temps limite, nous conservons la meilleure solution trouvée. Dans les tableaux suivants, le paramètre $MeilleureSolution$ indique le nombre de fois que cette méthode arrive à trouver la meilleure solution par rapport aux solutions trouvées par les autres méthodes avec qui elle est comparée.

Les différents tests réalisés et les résultats obtenus sont les suivants :

1. *Test 1* Comparaison entre les différents modes de comptage de divergences (*binaire* et *non-binaire*) et entre les trois grands principes d'exploration de l'arbre de recherche ; *LDS-prof*, *LDS-haut* (par divergences avec préférence au nœud qui se trouve au plus haut niveau), *LDS-bas* (par divergences avec préférence au nœud qui se trouve au plus bas niveau). Les recherches ont été réalisés sur les instances à 40 opérations (familles 3 et 4) et ce que nous avons évalué est la qualité des meilleures solutions trouvées et le nombre total de solutions explorées en moyenne (*Sol. explorées*) pour différentes recherches associées à différentes valeurs du nombre de divergences autorisées. Le temps de calcul est limité à 300 secondes. Nous avons cependant calculé le temps moyen T_{moyen} utilisé pour les différents méthodes, car certaines recherches s'achèvent en moins de 300 secondes.

Familles 3-4	Meilleure Solution	
(60 Instances)	Mode Binaire	Mode non-binaire
LDS-prof	52 (86.67 %)	11 (18.33 %)
LDS-haut	55 (91.67 %)	6 (10.00 %)
LDS-bas	55 (91.67 %)	8 (8.33 %)

Tab. 3. Résultats de la comparaison entre les modes de comptage des divergences

Le tableau 3 montre que le *mode binaire* est le plus performant pour ce type de problème. Avec ce mode, des meilleures solutions sont trouvées pour un pourcentage plus élevé d'instances qu'avec le *mode non-binaire*, et ce, indépendamment de la stratégie d'exploration utilisée.

D'autre part, en ce qui concerne la stratégie de branchement (voir tableau 4), *LDS-haut* donne de meilleurs résultats. Nous avons réalisé plusieurs recherches à différentes valeurs de divergences et cette stratégie permet de trouver plus rapidement des meilleures solutions ; la borne et le raisonnement énergétique augmentent leur efficacité. Avec *LDS-haut* la quantité de solutions explorées est considérablement plus petite qu'avec les deux autres méthodes.

Mode binaire	240 recherches		Familles 3 et 4
	Sol. explorées(moyen)	T_{moyen}	*Meilleure Solution*
LDS-prof	84474	137	23
LDS-haut	15880	125	50
LDS-bas	36494	131	29

Tab. 4. Résultats de la comparaison entre les stratégies de branchement pour un comptage binaire de divergences

2. *Test 2* Evaluation de la qualité du critère *ECT (Earliest Completion Time)* pour les problèmes intégrant contraintes de précédence et temps de préparation. Pour l'évaluation du critère, il nous faut connaître la solution optimale ; c'est la raison pour laquelle on utilise les familles de taille plus réduite, à 10 et à 15 opérations (familles 1 et 2).

   Nous avons considéré deux types d'instances différentes qui montrent des résultats sensiblement différents (voir tableau 5). Pour les instances avec un temps de préparation entre opérations supérieur au temps d'exécution, nous arrivons à trouver la solution optimale pour environ 80 % des instances en appliquant le critère ECT. D'autre part, pour les instances avec des temps de préparation et d'exécution du même ordre, ce pourcentage augmente jusqu'à 90 %.

   Ces résultats montrent que le critère *ECT* est très fiable, mais font aussi apparaître la nécessité d'envisager l'affectation des tâches sur les autres machines. Les pourcentages élevés de succès et la grande différence sur les temps de résolution indiquent que l'affectation des tâches sur les machines autres que celle qui permet de finir le plus tôt est très coûteuse en temps par rapport à la qualité des solutions obtenues. Nous proposons ainsi, lors de la recherche locale, la séparation pour le comptage des divergences en ce qui concerne l'affectation des opérations du comptage des divergences par rapport à la séquence des opérations.

	Instances	
	120 inst. p_i $[1-5]$ s_{ij} $[5-25]$	120 inst. p_i $[1-10]$ s_{ij} $[1-10]$
Solution Optimale avec ECT	95 (79,17 %)	109 (90.83 %)
T_{moyen} *ECT*	10.57	9.81
T_{moyen} Méthode Exacte	624.26	619.30

Tab. 5. Résultats de l'évaluation du critère *ECT*

3. *Test 3* Evaluation de la qualité de la borne, du raisonnement énergétique et des règles de dominance présentées. Nous avons réalisé 60 recherches à un nombre de divergences limité. Nous avons analysé leur effet sur la qualité de la solution obtenue et le temps nécessaire pour l'atteindre.

   Le tableau 6 montre que la borne inférieure et le raisonnement énergétique (*NRJ*) proposés sont efficaces, car leur application permet de réduire le temps de recherche et permet aussi de trouver de meilleures solutions dans les cas où la recherche n'est pas achevée. En ce qui concerne les règles de dominance, pour la règle de *Dominance de Front* (*DF*) on observe que son efficacité augmente avec le nombre de machines m. D'autre part, la règle de *Dominance Locale* (*DL*) pour l'affectation partielle donne des résultats très satisfaisants lorsqu'on l'applique avec la borne ou avec le raisonnement énergétique. Elle permet de réduire sensiblement le temps d'exploration et atteint la meilleure solution pour plus de 90 % des recherches réalisées.

(a)

$\sum C_i$	T_{moyen}	*Meilleure Solution*
Sans $lb(\sum C_i)$	258	45.00 %
$lb(\sum C_i)$	251	55.00 %
$lb(\sum C_i)$ + *DF*	247	55.00 %
$lb(\sum C_i)$ + *DL*	214	90.00 %
$lb(\sum C_i)$ + *DF* + *DL*	228	73.33 %

(b)

$L_{\max}$	T_{moyen}	*Meilleure Solution*
Sans *NRJ*	256	68.33 %
NRJ	250	86.67 %
NRJ + *DF*	248	86.67 %
NRJ + *DL*	232	93.33 %
NRJ + *DF* + *DL*	236	90.00 %

Tab. 6. Résultats de l'évaluation de la borne, du raisonnement énergétique et des règles de dominance

4. *Test 4* Comparaison entre *CDS (Climbing Discrepancy Search)*, la méthode *CDDS (Climbing Depth-bounded Discrepancy Search)* et les deux méthodes de recherche locale proposées *HD-CDDS* et *MC-CDS* implémentées avec *ILDS*, mode de comptage binaire et stratégie de branchement *LDS-haut*. Les problèmes sont à 100 opérations (familles 5 et 6) et le temps de calcul est limité à 300 secondes.

(a)

$\sum C_i$	*Meilleure Sol.*	*Ecart moyen*
CDS	12 (20.00 %)	11.93 %
CDDS	9 (15.00 %)	4.82 %
HD-CDDS	45 (75.00 %)	2.65 %
MC-CDS	1 (1.67 %)	9.77 %

(b)

$L_{\max}$	*Meilleure Sol.*	*Ecart moyen*
CDS	38 (63.33 %)	4.46 %
CDDS	19 (31.67 %)	9.46 %
HD-CDDS	39 (65.00 %)	4.53 %
MC-CDS	3 (5.00 %)	8.76 %

Tab. 7. Résultats de la comparaison entre les différentes méthodes de recherche locale

On observe tout d'abord que *HD-CDDS* est la plus performante. Elle présente les meilleures pourcentages en ce qui concerne le nombre de fois que la méthode atteint la meilleure solution et les plus petits écarts moyens par rapport aux meilleures solutions obtenues par les autres méthodes. Les résultats de la technique *CDDS* sont les moins bons pour ce type de problème. La cause en est que *CDDS* n'est pas une bonne technique de recherche locale lorsque des contraintes de précédence sont présentes dans le problème. On peut aussi remarquer que même si la meilleure solution est atteinte pour un pourcentage peu important pour *MC-CDS*, l'écart moyen par rapport aux meilleures solutions est acceptable (moins de 10 %).

A notre connaissance, il n'existe aucune méthode traitant le problème considéré dans la littérature pour une comparaison avec les méthodes que nous proposons. Aussi, nous proposons de valider ces méthodes hybrides de recherche locale basées sur les divergences par rapport à la meilleure solution connue sur les instances difficiles de problèmes à machines parallèles de type $P|r_i, q_i|C_{\max}$ (sans temps de préparation ni contraitnes de précédence) présentées dans [14] (50 instances avec $n = 50$ et $m = 7$ et 50 instances avec $n = 100$ et $m = 10$). La méthode *CDS* avec le calcul de bornes supérieures à chaque nœud arrive à trouver, sur les instances à $n = 50$, la meilleure solution connue pour 21 instances (4 instances parmi elles sont même améliorées). Pour les instances à 100 opérations, la meilleure solution connue est atteinte pour 35 instances et pour 6 d'entre elles cette meilleure solution est améliorée. Nous avons comparé aussi *CDDS*, avec une petite variante en ce qui concerne la limitation de profondeur, et la méthode est encore plus performante.

6 Conclusion

Dans cet article, nous proposons une analyse des méthodes de recherche arborescente à divergences limitées. Nous avons notamment utilisé ces techniques pour développer des méthodes de

recherche locale. Nous pouvons conclure que ces méthodes s'avèrent efficaces pour les problèmes à machines parallèles. Il faut aussi noter l'importance de l'emploi d'une technique de traitement des noeuds (calcul de borne inférieure ou coupe par raisonnement énergétique) adaptée au problème traité, de même que le paramétrage sur les divergences et profondeurs autorisées.

Nous avons aussi présenté deux conditions de dominance adaptées à la présence des divergences et une approche énergétique pour la minimisation du retard algébrique maximal, qui intègre les temps de préparation dans le calcul. Nous envisageons à présent une consolidation de la validation des méthodes presentées par une comparaison avec les résultats obtenus avec IlogScheduler.

Références

1. J. C. Beck and L. Perron. Discrepancy-bounded depth first search. *In Second International Workshop on Integration of AI and OR Technologies for Combinatorial Optimization Problems (CP-AI-OR'00)*, 2000.
2. Z.-L. Chen and W. B. Powell. Solving parallel machine scheduling problems by column generation. *INFORMS J. on Computing Vol :11,1 :78-94*, 1999.
3. T. Cheng and C. Sin. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research, Vol.47 :271-292*, 1990.
4. E. Demeulemeester, M. Vanhoucke, and W. Herroelen. Rangen : A random network generator for activity-on-the-node networks. *Journal of Scheduling 6 : 17-38*, 2003.
5. R.L Graham, E.L Lawler, J.K Lenstra, and A. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling :a survey. *Annals of Discrete Mathematics :287-326*, 1979.
6. W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. *In Proceedings of 14th IJCAI*, 1995.
7. A. Hmida, M. J. Huguet, P. Lopez, and M. Haouari. Climbing depth-bounded discrepancy search for solving hybrid flow shop scheduling problems. *European J. of Industrial Engineering 1, No.2 : 223 - 243*, 2007.
8. J. Hurink and S.Knust. List scheduling in a parallel machine environment with precedence constraints and setup times. *Operations Research Letters 29 : 231-239*, 2001.
9. W. Karoui, M.-J. Huguet, P. Lopez, and W. Naanaa. YIELDS : A yet improved limited discrepancy search for csps. *LNCS 4510, pp.99-111, Springer, 4th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'07)*, 2007.
10. R. Korf. Improved limited discrepancy search. *In Proceedings of 13th AAAI*, 1996.
11. P. Lopez. Approche par contraintes des problèmes d'ordonnancement et d'affectation : structures temporelles et mécanismes de propagation. *HDR, INP, Toulouse*, 2003.
12. M. Milano and A.Roli. On the relation between complete and incomplete search : an informal discussion. *In Proceedings CPAIOR'02*, 2002.
13. R. Nessah, Ch. Chu, and F. Yalaoui. An exact method for $Pm/sds, r_i/\sum C_i$ problem. *Computers and Operations Research 34 : 2840-2848*, 2005.
14. E. Néron, F. Tercinet, and F. Sourd. Search tree based approaches for parallel machine scheduling. *Computers and Operations Research*, 2006.
15. A. Salem, G. C. Anagnostopoulos, and G. Rabadi. A branch-and-bound algorithm for parallel machine scheduling problems. *Society for Computer Simulation International (SCS), Portofino, Italy, pp. 88-93*, 2000.
16. J.M.J. Schutten. List scheduling revisited. *Operations Research Letters 18, 167-170*, 1994.
17. T. Walsh. Depth-bounded discrepancy search. *APES Group, Department of Computer Science*, 1997.