7e Conférence Francophone de MOdélisation et SIMulation - MOSIM'08 - du 31 mars au 2 avril 2008 - Paris - France
«Modélisation, Optimisation et Simulation des Systèmes : Communication, Coopération et Coordination»

HEURISTIQUES POUR LE MEILLEUR DES CAS DANS UN ORDONNANCEMENT DE GROUPES

Guillaume Pinot, Nasser Mebarki

IRCCyN – UMR CNRS 6597
1, rue de la Noé – 44321 Nantes
guillaume.pinot@irccyn.ec-nantes.fr, nasser.mebarki@irccyn.ec-nantes.fr

RÉSUMÉ : *L'ordonnancement de groupes est une méthode d'ordonnancement du problème de* job shop *bien étudiée. Le but de cette méthode est de proposer de la flexibilité séquencielle pendant l'exécution de l'ordonnancement et de garantir une qualité minimale correspondant au pire des cas. Mais la qualité dans le meilleur des cas d'un ordonnancement de groupe serait également utile. En s'appuyant sur une relaxation de l'ordonnancement de groupe, cet article présente des heuristiques basées sur les règles de priorité et sur le* shifting bottleneck *pour ce problème. Les expérimentations effectuées sur ces heuristiques donnent de bonnes performances.*

MOTS-CLÉS : *Ordonnancement,* job shop*, ordonnancement de groupes, heuristiques,* shifting bottleneck*, ordonnancement sous incertitudes.*

1 INTRODUCTION

Le problème du *job shop* avec contraintes de précédence multiples ($J|r_i, \text{prec}|f$ d'après la classification de [Graham et al., 1979]) est un problème d'optimisation combinatoire composé de ressources, d'opérations et de contraintes. Les opérations (O_i) sont exécutées sur les ressources (M_i, aussi appelées machines) pendant un temps d'exécution p_i avec des contraintes de précédences (les prédécesseurs (resp. successeurs) de O_i sont données par $\Gamma^-(i)$ (resp. $\Gamma^+(i)$)). Une ressource ne peut exécuter qu'une opération à la fois. Une opération a une date de disponibilité r_i, sa date de début est notée t_i, et sa date de fin est notée C_i.

Généralement, le *job shop* utilise une fonction objectif régulière qui est une fonction monotone des C_i. Le but est de minimiser cette fonction objectif. Le *makespan*, noté $C_{\max}$, calculé $\max C_i$, qui correspond au temps total de l'exécution de l'ordonnancement, est un objectif régulier classique.

En pratique, les problèmes d'ordonnancement sont souvent soumis à une incertitude sur les données : incertitude sur les durées et les dates de disponibilité des opérations, opérations urgentes ou imprévues à réaliser, disponibilité incertaine des ressources [Artigues, 2004]. Pour pallier ces incertitudes, une méthode d'ordonnancement reposant sur l'introduction de flexibilité séquentielle dans les opérations a été proposée [Erschler and Roubellat, 1989]. Ceci est réalisé en déterminant, plutôt qu'un ordonnancement unique, un ordonnancement de groupes, c'est-à-dire en définissant une séquence de groupes d'opérations permutables sur chaque machine. L'évaluation d'un ordonnancement de groupes s'obtient par rapport à la qualité dans le pire des cas de l'ensemble d'ordonnancement réalisable.

Mais la qualité dans le meilleur des cas d'un ordonnancement de groupes peut également être intéressante pour différentes raisons. Elle donne des informations sur l'ordonnancement avant son exécution. Elle peut également être utile pour évaluer une décision pendant l'exécution de l'ordonnancement.

Dans cet article, après une présentation succincte de l'ordonnancement de groupes, nous présentons rapidement la méthode permettant de déterminer une relaxation de l'ordonnancement de groupe. Cette relaxation est par ailleurs décrite dans [Pinot and Mebarki, 2008]. Cette relaxation est ensuite utilisée dans des heuristiques d'ordonnancement classiques, afin d'améliorer leurs performances. Nous décrivons ces différentes heuristiques basées sur les règles de priorité et sur la méthode du *shifting bottleneck*. Enfin, la dernière partie de cet article traite des expérimentations réalisées sur un ensemble d'instance très utilisé dans la littérature. Ces expérimentations montrent des résultats prometteurs.

2 LES GROUPES D'OPÉRATIONS PERMUTABLES

On trouve dans [Erschler and Roubellat, 1989] une première présentation des groupes d'opérations permutables. Le but de cette méthode est de fournir de la flexibilité séquentielle pendant l'exécution de l'ordonnancement ainsi que de garantir une qualité minimale correspondant au pire des cas. Cette méthode est largement étudiée depuis vingt ans, notamment dans [Erschler and Roubellat, 1989, Billaut and Roubellat, 1996, Wu et al., 1999, Artigues et al., 2005]. Ce dernier article fournit une description théorique de la méthode.

Un groupe d'opérations permutables est un ensemble d'opérations à exécuter sur une certaine ressource dans

un ordre arbitraire. Il est noté G_k. Le groupe contenant l'opération O_i est noté $g(i)$.

Un ordonnancement de groupes est une liste ordonnée de groupes (d'opérations permutables) sur chaque machine, à exécuter dans cet ordre. Sur une machine, le groupe après (resp. avant) G_k est noté G_k^+ (resp. G_k^-) et est appelé le groupe successeur (resp. prédécesseur) de G_k. En utilisant les mêmes conventions, le groupe après (resp. avant) $g(i)$ est noté $g^+(i)$ (resp. $g^-(i)$).

Un ordonnancement de groupes est réalisable si chaque permutation sur les opérations d'un même groupe donne un ordonnancement réalisable (c'est-à-dire un ordonnancement qui ne viole pas de contraintes). Ainsi, un ordonnancement de groupes décrit un ensemble d'ordonnancement valide, sans les énumérer.

Les descripteurs de qualité d'un ordonnancement de groupes sont les même que pour un ordonnancement classique. Ainsi, la qualité d'un ordonnancement de groupes correspond à la pire qualité présente dans l'ensemble des ordonnancements semi-actifs, comme défini dans [Artigues et al., 2005].

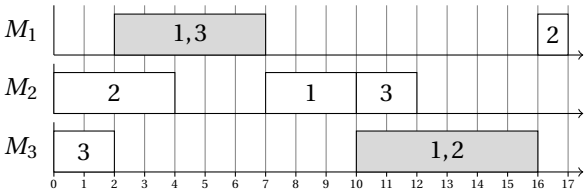
Pour illustrer ces définitions, étudions un exemple. Pour simplifier la lecture, nous utilisons ici le *job shop* sans contraintes de précédence multiples ($J||f$ d'après la classification de [Graham et al., 1979]). Les opérations sont indexées par un couple ($O_{i,j}$) et les opérations doivent être exécutées dans l'ordre du deuxième indice ($C_{i,j} \leq t_{i,j+1}$). Sur les figures, nous ne mettons que le premier indice pour faciliter la lisibilité. La figure 1b représente un ordonnancement de groupes admissible résolvant le problème de la figure 1a. Il est constitué de sept groupes : deux groupes de deux opérations et cinq groupes d'une opération. Cet ordonnancement de groupes définit quatre ordonnancements semi-actifs[1] correspondants (figure 2). On remarquera que ces ordonnancements ont des qualités différentes : la qualité dans le meilleur des cas est $C_{\max} = 10$ et la qualité dans le pire des cas est $C_{\max} = 17$.

L'ordonnancement de groupes possède une propriété intéressante : le calcul de la qualité d'un ordonnancement de groupes dans le pire des cas se fait en temps polynomial (voir [Artigues et al., 2005] pour la description de l'algorithme). Ainsi, il est possible de calculer sans problème la qualité dans le pire des cas, même pour les ordonnancement de groupes de très grande taille. Par conséquent, cette méthode peut être utilisée pour calculer la qualité dans le pire des cas en temps réel pendant l'exécution de l'ordonnancement. Grâce à cette propriété, il est possible de surveiller la qualité d'un ordonnancement de groupes dans un système d'aide à la

[1] Un ordonnancement semi-actif est un ordonnancement où l'on exécute les opérations au plus tôt. Pour une séquence d'opérations, il existe un unique ordonnancement semi-actif.

i	j	$M_{i,j}$	$p_{i,j}$
1	1	1	3
1	2	2	3
1	3	3	3
2	1	2	4
2	2	3	3
2	3	1	1
3	1	3	2
3	2	1	2
3	3	2	2

(a) Un problème de *job shop*



(b) Un ordonnancement de groupe résolvant le problème décrit dans la figure 1a

FIG. 1: Un problème de *job shop* résolu par un ordonnancement de groupes

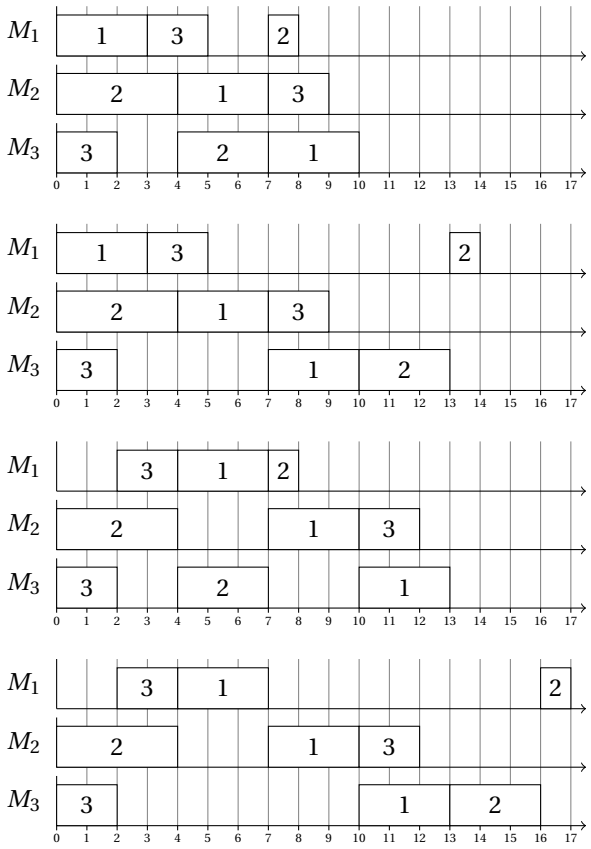


FIG. 2: Ordonnancements semi-actifs décrits par la figure 1b

décision de manière dynamique.

Cette méthode permet donc de décrire un ensemble d'ordonnancement de manière implicite (c'est-à-dire sans énumérer les ordonnancements) tout en garantissant une certaine qualité. Lors de l'exécution d'un tel ordonnancement, il est possible de choisir la séquence d'opérations adapté à l'état réel de l'atelier.

De plus, la flexibilité ajoutée à l'ordonnancement devrait permettre d'absorber des incertitudes. Trois études ont essayé de vérifier cette propriété.

[Wu et al., 1999] étudie l'impact de la perturbation des temps d'exécution sur la somme pondérée des retards par rapport à des heuristiques statiques et dynamiques. Quand les temps d'exécutions ne sont pas trop perturbés, l'ordonnancement de groupes donne de meilleures performances.

[Esswein, 2003] étudie l'impact de la perturbation des temps d'exécutions, des dates de livraison et des dates de disponibilité sur un problème à une machine, et fait des comparaisons avec une heuristique statique. En moyenne, les performances sont meilleures avec l'ordonnancement de groupes.

[Pinot et al., 2007] étudie l'impact sur un système réel de la non modélisation de temps de transport entre deux opérations dans l'ordonnancement de groupes. La méthode montre de bonnes performances, même quand les temps de transport du système réel sont comparables aux temps d'exécution.

3 UNE RELAXATION DE L'ORDONNANCEMENT DE GROUPES

3.1 Le *one-machine problem*, relaxation du *job shop*

Dans le problème de *job shop* avec comme critère le *makespan*, des outils très utiles sont les têtes (*heads*) et les queues (*tails*) des opérations. La tête d'une opération correspond à une borne inférieure de sa date de début au plus tôt, alors que la queue d'une opération correspond à une borne inférieure au temps minimal entre la fin de l'opération et la fin de l'ordonnancement.

Ces outils sont généralement utilisés pour la relaxation du *job shop* en *one-machine problem* [Carlier, 1982]. Cette relaxation consiste à ne considérer qu'une machine, en utilisant les têtes comme des dates de disponibilités et les queues comme des temps de travail à effectuer après l'opération, le but étant d'optimiser le *makespan*. Ce problème est équivalent à un problème à une machine avec dates de disponibilité dont le but est de minimiser le retard algébrique maximum (soit le problème $1|r_i|L_{\max}$). Ce problème est NP-difficile, mais [Carlier, 1982] fournit une méthode exacte efficace.

Cette relaxation est utilisée dans le calcul des bornes inférieures du *job shop* [Carlier and Pinson, 1989, Carlier and Pinson, 1990, Carlier and Pinson, 1994, Brucker et al., 1994] ainsi que pour le calcul d'heuristiques comme le *shifting bottleneck* [Adams et al., 1988] et les heuristiques utilisées dans [Brucker et al., 1994]. La qualité des têtes et queues sont des facteurs cruciaux pour la performance de ces bornes et heuristiques.

Obtenir une relaxation de l'ordonnancement de groupes en *one-machine problem* serait un outil précieux pour la résolution de l'ordonnancement de groupe dans le meilleur des cas. C'est ce que nous proposons dans la suite de cette section.

3.2 Têtes pour l'ordonnancement de groupes

Notre but dans cette section est de calculer la date de début d'une opération dans le meilleur des cas, qui correspond à la plus petite valeur de t_i dans chaque ordonnancement semi-actif décrit par un ordonnancement de groupes. Comme ce problème est NP-difficile, il peut être très utile d'obtenir une borne inférieure sur ces dates de début dans le meilleur des cas en temps polynomial. Une telle borne inférieure serait une tête valide de l'opération O_i.

Il est facile de calculer une borne inférieure à notre problème en utilisant une relaxation sur les ressources (C'est-à-dire en considérant les ressources comme des ressources à capacité infinie). Dans ce cas, la borne inférieure de la date de début d'une opération dans le meilleur des cas (θ_i) est calculée comme le maximum (des bornes inférieures) des dates de fin au plus tôt (χ_i) de tous ses prédécesseurs. Pour l'opération O_i, les prédécesseurs correspondent aux prédécesseurs donnés par le problème ($\Gamma^-(i)$) mais également aux opérations du groupe prédécesseur (les opérations du groupe $g^-(i)$). Par exemple, dans l'exemple décrit figure 1, les prédécesseurs de l'opération $O_{2,3}$ (exécuté sur M_1) sont l'opération $O_{2,2}$ (exécutée sur M_3) à cause de la contrainte de précédence, et les opérations $O_{1,1}$ et $O_{3,2}$ (exécutées sur la même machine M_1) car elles sont dans le groupe prédécesseur ($g^-(2,3)$). Nous avons donc :

$$\begin{cases} \theta_i = \max\left(r_i, \max\limits_{j\in g^-(i)} \chi_j, \max\limits_{j\in\Gamma^-(i)} \chi_j\right) \\ \chi_i = \theta_i + p_i \end{cases} \quad (1)$$

Calculer θ_i en utilisant (1) est équivalent au calcul de la tête de l'opération O_i comme décrit dans [Carlier and Pinson, 1989].

Mais cette borne peut être améliorée grâce à une propriété de l'ordonnancement de groupes : une opération dans un groupe donné ne peut être exécutée avant que toutes les opérations du groupe prédécesseur ne soient exécutées. Ainsi, une opération ne peut commencer qu'après la plus petite date de fin du groupe prédé-

cesseur.

Il est donc nécessaire de calculer la plus petite date de fin d'un groupe ou une borne inférieure de cette date. Nous avons précédemment calculé les θ_i du groupe, qui sont des bornes inférieures aux dates de disponibilité des opérations. Pour calculer une borne inférieure de la plus petite date de fin d'un groupe, nous pouvons générer un problème du type $1|r_i|C_{\max}$ qui correspond à notre problème, avec $r_i = \theta_i$. Ce problème est résolu en temps polynomial en ordonnant les opérations par ordre croissant de date de disponibilité [Brucker and Knust, 2007, Lawler, 1973].

Ainsi, nous pouvons calculer une borne inférieure de la date de début d'une opération dans le meilleur des cas (θ_i), une borne inférieure de la date de fin d'une opération dans le meilleur des cas (χ_i), et une borne inférieure de la date de fin d'un groupe dans le meilleur des cas (γ_i) :

$$\begin{cases} \theta_i = \max\left(r_i, \gamma_{g^-(i)}, \max\limits_{j\in\Gamma^-(i)} \chi_j\right) \\ \chi_i = \theta_i + p_i \\ \gamma_k = C_{\max} \text{ de } 1|r_i|C_{\max}, \forall O_i \in G_k, r_i = \theta_i \end{cases} \quad (2)$$

3.3 Exemple de l'exécution de l'algorithme

Voyons un exemple pour illustrer ces algorithmes. Pour garder cet exemple compréhensible, nous utiliserons un problème simple de type *flow shop*. Le problème, l'ordonnancement de groupes ainsi que les valeurs calculées par les algorithmes sont présentés dans le tableau 1. Une représentation dans le pire des cas de l'ordonnancement de groupes est disponible figure 3. Les quatre ordonnancements semi-actifs décrits par l'ordonnancement de groupes sont représentés sur la figure 4. Ils sont utilisés pour calculer la valeur optimale dans le tableau 1. La figure 5 est une représentation sous forme de graphe du calcul de (2), qui montre les dépendances entre les différentes entités.

Nous pouvons remarquer l'amélioration de (2) par rapport à (1) pour l'opération $O_{3,1}$. Le calcul du *makespan* du groupe G_1 améliore la borne inférieure $\theta_{3,1}$ jusqu'à sa valeur optimale.

L'équation (2) ne donne pas la valeur optimale pour $\theta_{3,2}$: la valeur optimale est 6, mais (2) trouve 5. Cette approximation est causée par le fait que deux dates de début dans le meilleur des cas n'apparaissent pas forcément dans le même ordonnancement semi-actif. Dans notre exemple, il ne peut y avoir d'ordonnancement semi-actif avec $t_{1,2} = \theta_{1,2} = 1$ (premier ordonnancement de la figure 3) et $t_{2,2} = \theta_{2,2} = 3$ (quatrième ordonnancement de la figure 3). En effet si $t_{1,2} = 1$, alors $t_{1,2}$ ne peut être inférieur à 4 à cause des contraintes de précédences. Ainsi, γ_{G_2} n'est pas égale au *makespan* optimal

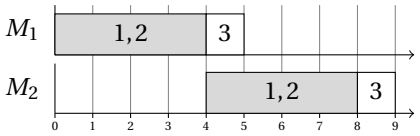


FIG. 3: L'ordonnancement de groupes décrit par le tableau. 1

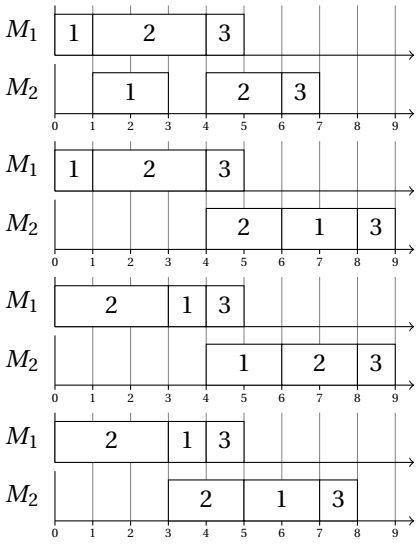


FIG. 4: Les ordonnancements semi-actifs décrits par la figure 3

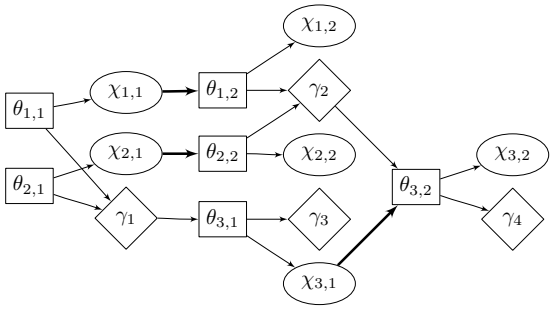


FIG. 5: Une représentation sous forme de graphe de l'exécution de (2) pour le problème présenté dans le tableau 1

Description du problème					Équation (1)		Équation (2)			Val. optimale	
i	j	$M_{i,j}$	$p_{i,j}$	G_k	$\theta_{i,j}$	$\chi_{i,j}$	$\theta_{i,j}$	$\chi_{i,j}$	$\gamma_{g(i,j)}$	$\theta_{i,j}$	$\chi_{i,j}$
1	1	1	1	G_1	0	1	0	1	4	0	1
1	2	2	2	G_2	1	3	1	3	5	1	3
2	1	1	3	G_1	0	3	0	3	4	0	3
2	2	2	2	G_2	3	5	3	5	5	3	5
3	1	1	1	G_3	3	4	**4**	**5**	5	4	5
3	2	2	1	G_4	5	6	5	6	6	**6**	**7**

TAB. 1: Exemple d'un *flow shop*

du groupe G_2 (6, comme dans le premier ordonnancement de la figure 3), mais à une borne inférieure (5). Par la suite, l'erreur se propage aux groupes successeurs.

3.4 Relaxation de l'ordonnancement de groupe en *one-machine problem*

Nous possédons déjà une tête pour l'ordonnancement de groupes : θ_i en utilisant (2). Pour obtenir une queue, nous pouvons renverser (2), c'est-à-dire plutôt que de commencer le calcul au début de l'ordonnancement, le commencer par la fin. Ainsi, en remplaçant les prédécesseurs par les successeurs, nous obtenons :

$$\begin{cases} \theta'_i = \max\left(\gamma'_{g^+(i)}, \max\limits_{j\in\Gamma^+(i)} \chi'_j\right) \\ \chi'_i = \theta'_i + p_i \\ \gamma'_k = C_{\max} \text{ de } 1|r_i|C_{\max}, \forall O_i \in G_k, r_i = \theta'_i \end{cases} \quad (3)$$

avec θ'_i une queue valide.

Pour l'ordonnancement de groupes, la relaxation se fera naturellement au niveau des groupes plutôt qu'au niveau des machines (comme c'est le cas dans le problème du *job shop*). Les problèmes seront alors plus petits, ce qui facilitera leur résolution exacte.

Cette relaxation peut être utilisée pour la réalisation d'heuristiques. C'est ce que nous décrivons dans la section suivante.

4 HEURISTIQUES À BASE DE RÈGLES DE PRIORITÉ

Les heuristiques basées sur les règles de priorité des files d'attente sont parmi les méthodes les plus simples et les plus utilisées en pratique pour ordonnancer un atelier de type job-shop. [Blackstone et al., 1982] définit une règle de priorité comme une méthode permettant de sélectionner parmi tous les travaux en attente de traitement sur une ressource, le prochain travail à traiter, selon une priorité prédéfinie.

4.1 Règles de priorité

La règle de priorité *most work remaining*, qui correspond à exécuter le travail possédant le plus long temps d'exécution restant, est la règle de priorité classique la plus efficace pour le *makespan*. C'est pourquoi nous prenons cette règle comme heuristique de référence. Cette heuristique sera par la suite nommée MWR.

Nous proposons une règle de priorité basée sur la queue de l'opération. L'idée est de donner une priorité importante à une opération possédant une grande queue, c'est-à-dire une opération risquant de perturber le *makespan*. Comme les queues des opérations sont parfois égales, la règle est combinée avec *Smallest Processing Time* (SPT). La règle s'écrit donc ainsi :

$$\min_{\forall O_i} p_i - {\theta'_i}^2$$

Cette heuristique sera par la suite nommée SQUTAIL.

4.2 Utilisation d'une règle de priorité avec une borne inférieure

Dans [Pinot and Mebarki, 2008], nous définissons une borne inférieure efficace pour le *makespan* en utilisant la méthode décrite dans la section 3. Un ordonnancement partiel possédant une borne inférieure plus petite donnera généralement un ordonnancement final de meilleure qualité. Pour cette raison, l'utilisation d'une telle borne inférieure dans une règle de priorité semble judicieuse.

Cette règle de priorité est définie de la façon suivante :

1. pour chaque opération présente dans la file d'attente, on génère un ordonnancement (de groupes) partiel où cette opération est sélectionnée immédiatement ;
2. on calcule la borne inférieure pour ces ordonnancements partiels générés ;
3. on sélectionne l'opération ayant abouti à la plus petite borne inférieure.

Néanmoins, cette règle de priorité conduit à de nombreuses égalités de priorité (*i.e.* des opérations donnat la même borne inférieure). L'utilisation de la borne inférieure seule n'est donc pas suffisante.

C'est pourquoi nous proposons de combiner la borne inférieure avec une autre règle de priorité. Dans un premier temps, les opérations dont l'exécution donne une

borne inférieure minimale sont sélectionnées. Dans un second temps, l'opération à exécuter est sélectionnée par une règle de priorité parmi les opérations sélectionnées précédemment.

Les règles de priorité utilisées sont celles décrite dans la section 4.1. L'utilisation de MWR avec la borne inférieure sera par la suite nommée LB+MWR (LB pour *lower bound*). De la même façon, l'utilisation de SQUTAIL avec la borne inférieure sera nommée LB+SQUTAIL

5 UN *SHIFTING BOTTLENECK* POUR L'ORDONNANCEMENT DE GROUPES

5.1 Le *shifting bottleneck*

Le *shifting bottleneck*, décrit dans [Adams et al., 1988], est une heuristique très connue et efficace pour le problème de *job shop* avec comme objectif le *makespan*.

Nous présentons ici une idée globale de l'algorithme. Pour une explication détaillée de l'algorithme, voir [Adams et al., 1988].

À chaque itération, la machine goulot est sélectionnée, l'ordre d'exécution des opérations sur cette machine est alors fixé. Les machines déjà séquencées sont ensuite réoptimisées. Cette procédure est répétée jusqu'à ce que toutes les machines soient séquencées. On obtient alors l'ordonnancement.

Pour le choix de la machine à séquencer, ainsi que pour son séquencement, la relaxation en *one-machine problem* est utilisée. L'algorithme de Carlier [Carlier, 1982] est utilisé pour séquencer les machines.

Théoriquement, cet algorithme peut donner des ordonnancements non réalisables. En effet, la relaxation en *one-machine problem* ne prend pas en compte les contraintes de précédence. L'algorithme peut donc donner un ordonnancement violant des contraintes, ce qui donnerait un ordonnancement non réalisable. Pour pallier ce défaut, les auteurs traitent ce cas rare comme une exception.

5.2 Adaptation pour l'ordonnancement de groupes

Pour réaliser un *shifting bottleneck* pour l'ordonnancement de groupes, nous pouvons utiliser la relaxation en *one-machine problem* décrit section 3. L'algorithme n'est alors plus effectué au niveau des machines mais au niveau des groupes. Le nombre de réoptimisations est alors beaucoup plus important que dans le *shifting bottleneck* classique. Nous nous attendons donc à de meilleures performances, mais également à un temps de calcul plus grand.

La relaxation au niveau des groupes permet d'éviter le risque de cycle présent dans l'algorithme original : comme le choix des séquences est effectué au niveau des groupes, et que toutes les permutations des opérations dans un groupe donne par définition un ordonnancement valide, il ne peut y avoir de violation de contrainte.

Cette heuristique sera par la suite nommée SB.

Dans la section suivante, nous comparons ces heuristiques sur un ensemble d'instances très utilisé dans la litterature du *job shop*.

6 EXPÉRIMENTATION

6.1 Protocole

Le but de ces expérimentations est de comparer les différentes heuristiques présentées dans cet article pour l'ordonnancement de groupes.

Nous utilisons un ensemble d'instance très utilisée nommé `la01` à `la40` de [Lawrence, 1984]. Ce sont des instances de *job shop* classiques, avec m opérations pour chaque travail (m le nombre de machines), chaque opération d'un travail s'exécutant sur une machine différente. Cet ensemble est composé de 40 instances de tailles différentes (5 instances pour chaque taille).

Pour chaque instance, nous générons un ordonnancement de groupes avec une qualité optimale connue et une très grande flexibilité. Pour générer ces ordonnancements, nous utilisons un algorithme glouton qui fusionne deux groupes successeurs en fonction de différents critères jusqu'à ce qu'il n'y ait plus de fusion de groupes possible. Cet algorithme commence avec un ordonnancement de groupes à une opération par groupe calculé par l'algorithme exact décrit dans [Brucker et al., 1994] (donc, par construction, le *makespan* optimal de ces ordonnancements de groupes est le *makespan* de cet ordonnancement de groupes à une opération par groupe). L'algorithme glouton est décrit dans [Esswein, 2003]. Le code source du programme utilisé pour réaliser ces expérimentations peut être téléchargé à `http://www.irccyn.ec-nantes.fr/~pinot/`.

Pour chaque ordonnancement de groupes, nous calculons l'écart entre le *makespan* de chaque heuristique et le *makespan* optimal de l'ordonnancement de groupes. Les résultats sont présentés sous la forme de boîtes à moustache (*boxplots*) dans la figure 6. Les résultats et les temps d'exécutions correspondant aux tailles des instances sont présentés dans le tableau 2.

	MWR		SQUTAIL		LB+MWR		LB+SQUTAIL		SB	
Taille	Écart	Tps (s)	Écart	Tps (s)	Écart	Tps (s)	Écart	Tps (s)	Écart	Tps (s)
10 × 5	0,3014	0,0005	0,0905	0,0072	0,0302	0,0542	0,0385	0,0542	0,0211	0,0524
15 × 5	0,0972	0,0007	0,0075	0,0202	0,0136	0,1737	0,0124	0,1714	0,0046	0,1259
20 × 5	0,1347	0,0010	0,0000	0,0399	0,0030	0,3392	0,0017	0,3373	0,0000	0,2587
10 × 10	0,4509	0,0019	0,2956	0,0322	0,0374	0,1866	0,0318	0,1861	0,0182	0,8292
15 × 10	0,2941	0,0022	0,1807	0,0862	0,0426	0,5511	0,0426	0,5506	0,0197	2,2478
20 × 10	0,3094	0,0029	0,1356	0,1720	0,0410	1,1926	0,0668	1,1888	0,0201	4,6298
30 × 10	0,1847	0,0050	0,0616	0,4559	0,0362	3,8018	0,0269	3,7752	0,0000	10,5885
15 × 15	0,6441	0,0065	0,3237	0,1996	0,0758	1,2366	0,0749	1,2273	0,0321	10,7351
Moyenne	0,3021	0,0026	0,1369	0,1266	0,0350	0,9420	0,0369	0,9364	0,0145	3,6834

La taille est notée $n \times m$ avec n le nombre de travaux et m le nombre de machines.

TAB. 2: Écarts moyens et temps d'exécution des différentes heuristiques par rapport à la taille du problème

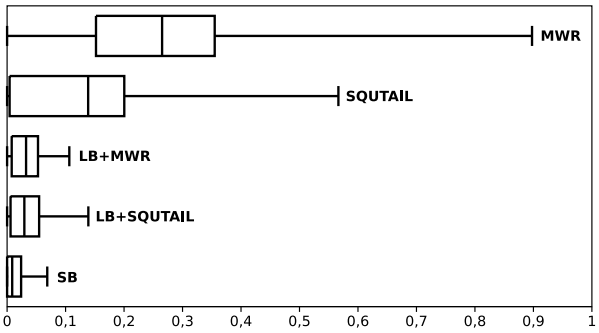


FIG. 6: Écarts des heuristiques par rapport à l'optimal

6.2 Discussion

MWR possède des performances décevantes par rapport aux autres heuristiques avec un écart moyen à 0,3. Elle est par contre très rapide.

SQUTAIL est bien meilleure que MWR avec une moyenne à 0,14. Par contre, bien qu'elle ne soit qu'une simple règle de priorité, elle est beaucoup plus coûteuse que MWR. En effet, l'équation 3 permettant de calculer θ'_i doit être exécutée à chaque décision, ce qui donne beaucoup de calculs supplémentaires.

LB+MWR et LB+SQUTAIL ont des performances et des temps d'exécution comparables. La borne inférieure seule comme décrit section 4.2 serait donc une bonne règle de priorité. Par contre, la règle de priorité associée à la borne inférieure semble avoir moins d'impact sur les résultats, bien que LB+MWR soit légèrement meilleure que LB+SQUTAIL. Ceci est peut-être du au fait que la borne inférieure utilise abondamment les queues θ'_i, et donc MWR apporterait une information supplémentaire. Les deux heuristiques ont des temps de calcul équivalent, mais bien supérieurs aux temps d'exécution de SQUTAIL et MWR. En effet, la borne inférieure doit être calculée pour chaque opération dans la file d'attente.

SB est de loin la meilleure heuristique. Avec 17 valeurs optimales trouvées sur 40 et un écart moyen de 0,015, ses performances sont plus de deux fois supérieures à celles de LB+MWR et LB+SQUTAIL et 50 fois supérieures à celle de SQUTAIL. Son temps d'exécution est bien sûr également plus important : elle est environ 4 fois plus coûteuse que LB+SQUTAIL.

Globalement, on remarque que les instances longues (*i.e.* les instances possédants un rapport n/m élevé, n le nombre de travaux, m le nombre de machines) donnent des écarts faibles. Cela est cohérent avec le fait que les instances longues de *job shop* semblent être plus simples dans [Brucker et al., 1994].

7 CONCLUSION

Dans cet article, nous avons présenté une méthode de relaxation de l'ordonnancement de groupe adapté au *makespan* dans le meilleur des cas. Cette relaxation est combinée à des heuristiques d'ordonnancement très utilisées comme la méthode du *shifting bottleneck*. Les expérimentations réalisées en utilisant ces nouvelles heuristiques présentent de très bons résultats, notamment l'adaptation du *shifting bottleneck* qui a en moyenne un écart à l'optimal de 1,5%.

Après les bornes inférieures proposées par [Pinot and Mebarki, 2008] et les heuristiques proposées par cet article, il serait intéressant de proposer des méthodes exactes pour calculer la qualité dans le meilleur des cas.

REFERENCES

[Adams et al., 1988] Adams, J., Balas, E., and Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3) :391–401.

[Artigues, 2004] Artigues, C. (2004). Optimisation et robustesse en ordonnancement sous contraintes de ressources. Habilitation à diriger des recherches, Université d'Avignon.

[Artigues et al., 2005] Artigues, C., Billaut, J.-C., and Esswein, C. (2005). Maximization of solution flexibility for robust shop scheduling. *European Journal of Operational Research*, 165(2) :314–328.

[Billaut and Roubellat, 1996] Billaut, J.-C. and Roubellat, F. (1996). A new method for workshop real-time scheduling. *International Journal of Production Research*, 34(6) :1555–1579.

[Blackstone et al., 1982] Blackstone, J. H., Phillips, D. H., and Hogg, G. L. (1982). A state of the art survey of dispatching rules for manufacturing job shop operations. *International Journal of Production Research*, 20 :27–45.

[Brucker et al., 1994] Brucker, P., Jurisch, B., and Sievers, B. (1994). A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49(1-3) :107–127.

[Brucker and Knust, 2007] Brucker, P. and Knust, S. (2007). Complexity results for scheduling problems. `http://www.mathematik.uni-osnabrueck.de/research/OR/class/`. [online ; retrieved on 2007-05-21].

[Carlier, 1982] Carlier, J. (1982). The one-machine sequencing problem. *European Journal of Operational Research*, 11(1) :42–47.

[Carlier and Pinson, 1989] Carlier, J. and Pinson, E. (1989). An algorithm for solving the job-shop problem. *Management Science*, 35(2) :164–176.

[Carlier and Pinson, 1990] Carlier, J. and Pinson, E. (1990). A practical use of jackson's preemptive schedule for solving the job shop problem. *Annals of Operations Research*, 26(1-4) :269–287.

[Carlier and Pinson, 1994] Carlier, J. and Pinson, E. (1994). Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78(2) :42–47.

[Erschler and Roubellat, 1989] Erschler, J. and Roubellat, F. (1989). An approach for real time scheduling for activities with time and resource constraints. In Slowinski, R. and Weglarz, J., editors, *Advances in project scheduling*. Elsevier.

[Esswein, 2003] Esswein, C. (2003). *Un apport de flexibilité séquentielle pour l'ordonnancement robuste*. Thèse de doctorat, Université François Rabelais Tours.

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

[Lawler, 1973] Lawler, E. L. (1973). Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19(5) :544–546.

[Lawrence, 1984] Lawrence, S. (1984). Resource constrained project scheduling : an experimental investigation of heuristic scheduling techniques (supplement). Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

[Pinot et al., 2007] Pinot, G., Cardin, O., and Mebarki, N. (2007). A study on the group sequencing method in regards with transportation in an industrial FMS. In *Proceedings of the IEEE SMC 2007 International Conference*.

[Pinot and Mebarki, 2008] Pinot, G. and Mebarki, N. (2008). Best-case lower bounds in a group sequence for the job shop problem. In *Proceedings of the 17th IFAC World Congress*.

[Wu et al., 1999] Wu, S. D., Byeon, E.-S., and Storer, R. H. (1999). A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness. *Operations Research*, 47(1) :113–124.