Job-shop de permutation ?

Pierre Baptiste[1], Randa Ouchène[2], Djamal Rebaine[2]
[1] Polytechnique Montréal, Canada
Pierre.baptiste@polymtl.ca
[2] Université du Québec À Chicoutimi, Canada
rouchene@etu.uqac.ca; Djamal_rebaine@uqac.ca

Mots-clés*: job-shop, permutation, makespan.*

1 Introduction

Dans l'univers ordonnancement, « permutation » est associé au flow-shop, défini habituellement par « le même ordre de passage sur toutes les machines » ; Pinedo [6], lui, le définit par « FIFO sur chaque machine », ce qui est équivalent dans le cas du flow-shop simple. Réduire l'espace de recherche des solutions à un seul ordre simplifie les calculs, la règle FIFO (First In First Out) est la plus facile à implanter sur le terrain et, dans les services, elle est la plus acceptable (qui aime se faire doubler ?). Cette définition a l'avantage de s'appliquer à une plus grande variété de problèmes, car elle ne suppose pas que l'ensemble de jobs utilise toutes des machines : flow-shop hybride ou flexible, flow-shop avec opérations manquantes, etc. Elle peut aussi s'étendre au-delà du flow-shop, par exemple au job-shop.

2 Comparaison du J//Cmax et du J/FIFO/Cmax (job-shop de permutation)

La contrainte FIFO a été étudiée comme une des règles de priorité dans le cas d'ordonnancement temps réel [4], mais ne semble pas avoir été étudiée dans le cas d'optimisation avec des modèles mathématiques. Le but de cette recherche est d'évaluer la détérioration du Cmax induite en imposant FIFO sur toutes les machines, dans job-shop. La méthodologie de recherche a consisté à choisir un modèle mathématique pour le job-shop J//Cmax, à y intégrer la contrainte FIFO, puis à comparer les Cmax résultant sur des benchmarks et des problèmes aléatoires.

Le modèle utilisé est simple : tâches $O_{i,j}$ indexées par le job i et la machine j et leur durée $p_{i,j}$; une variable $R_{i,j}$ donnant le rang l'opération $O_{i,j}$; une contrainte d'antériorité des tâches d'un job et une de non-chevauchement de deux jobs faits sur la même machine ; des contraintes de positivité.

Nous rajoutons une variable $PR_{i,j}$ donnant la machine sur laquelle se fait l'opération précédant une opération $O_{i,j}$ donnée.

Cette variable permet d'exprimer la date d'arrivée d'une opération dans une file (date de fin de l'opération précédente) et d'interdire qu'une date de début d'une opération puisse être inférieure à la date de début d'une opération arrivée plus tôt, ce qui rajoute N^2*M contraintes.

Pour la première opération de chaque job, la variable $PR_{i,j}$ est arbitrairement mise à « -1 » afin d'ignorer la contrainte FIFO pour la première opération.

2.1 Résultats sur les benchmarks

Les deux modèles $J||C_{max}$ et $J|fifo|C_{max}$ ont été testés sur 40 benchmarks de la littérature, 6*6, 10*5, 10*10 ; 15*10 et 15*15 ([2][3][4][5][7]). L'écart des solutions optimales est en moyenne de 1,433% et le max est de 3,52%, obtenu sur un petit problème (10 jobs 5 machines).

2.2 Résultats sur des cas générés aléatoirement

Nous avons généré 100 instances de problèmes 10*10, avec des durées uniformément distribuées (de 1 à 100) et un ordre de passage aléatoire. La figure (1) montre que l'écart entre « avec » ou « sans FIFO » est faible (moyenne 1,16 ; écart type 0,96 ; max 3,47 ; min 0) ; cela corrobore les écarts des benchmarks.

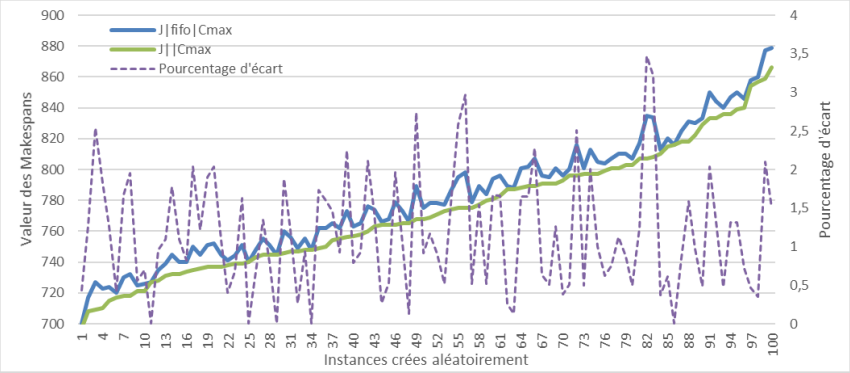

Figure 1 : Makespan et pourcentage d'écarts ses solutions optimales de $J||C_{max}$ (vert) et $J|fifo|C_{max}$ (bleu)

3 Conclusions

Le très faible écart entre les résultats avec ou sans FIFO donne à penser qu'elle pourrait facilement être imposée dans tous les cas où elle apporte une simplification dans la mise en œuvre d'une solution d'ordonnancement ou une meilleure acceptabilité des solutions.

Le modèle étudié ne considère pas tous les jobs disponibles à t=0 : il fournit une « *release date* » correspondant sa date d'entrée dans la file pour sa première opération. Il est donc particulièrement adapté à un système de prise de rendez-vous.

Comme pour tous les modèles mathématiques, et contrairement aux métaheuristiques, le modèle ne suppose pas que les ordonnancements soient sans délai : une machine peut arbitrairement attendre pour traiter le premier job de sa file afin qu'il « perde » une priorité possible sur une machine subséquente !

Finalement, pour des problèmes de grande taille, il pourrait être intéressant de chercher des métaheuristiques, mais l'espace de recherche est plus complexe qu'en flow-shop.

Références

1 Adams, J., Balas, E., Zawack, D., 1988. The shifting bottleneck procedure for job [shop scheduling. Management Science 34, 391–401.

2 Applegate, D., Cook, W., 1991. A computational study of job-shop scheduling. ORSA Journal of Computing 3, 149–156.

3 Fisher, H., Thompson, G.L., 1963. Probabilistic learning combinations of local job-shop scheduling rules, in: Muth, J., Thompson, G. (Eds.), Industrial Scheduling. Prentice Hall, pp. 225–251.

4 Haupt, R., 1989. A survey of priority rule-based scheduling. Operations-Research-Spektrum 11, 3–16.

5 Lawrence, S., 1984. Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement). Technical Report. Carnegie-Mellon University

6 Pinedo, M., 2002. Scheduling: Theory, Algorithms and Systems. Springer, New York, p. 15..

7 Taillard, E.D., 1993. Benchmarks for basic scheduling problems. European Journal of Operational Research 64, 278–285.