Ordonnancement d'un atelier de peinture avec LocalSolver

Fumio Kamijo[1] Thierry Benoist[2]

[1] 0-1 integer Incorporated,
Kawashima building at number 12 chome, Kanda Jimbo-cho, Chiyoda-ku, Tokyo
kamijo@0-1integer.co.jp

[2] LocalSolver
24, Avenue Hoche, 75008 Paris, France
tbenoist@localsolver.com

Mots-clés *: JobShop scheduling, Shortest Common Supersequence, LocalSolver*

1 Problématique industrielle

Dans un atelier de peinture, N objets doivent être peints avec de multiples couleurs dans un ordre donné. On dispose d'une unique machine et il faut minimiser le nombre de changements de couleur. En d'autres termes, N travaux doivent être exécutés sur une machine unique, chaque travail étant une séquence d'opérations de peinture (chacune identifiée par sa couleur). A chaque pas de temps, la machine peut accomplir un nombre illimité d'opérations pourvu que celles-ci soient de la même couleur. Cependant, elle ne peut pas traiter l'opération *o* du travail *j* tant que l'opération *o-1* de ce travail n'a pas été réalisée. L'objectif est de minimiser le nombre de pas de temps nécessaires pour terminer tous les travaux.

Dans la littérature, ce problème est connu comme le problème de la plus courte superséquence commune *(Shortest Common Supersequence)*. Etant donnés N mots, trouver le plus petit mot w tel que chacun de ces N mots puisse être obtenu en supprimant des lettres de w. Nous avons cependant conservé ici la formulation sous la forme de problème d'ordonnancement car c'est sous cette forme que la société japonaise *0-1 integer* (http://0-1integer.co.jp/) a rencontré cette problématique.

2 Modélisation

Nous considérons ici une résolution par LocalSolver [1], solveur de programmation mathématique à base de recherche locale, que nous comparerons avec les meilleurs résultats obtenus sur ce problème par des approches à base de programmation linéaire en nombres entiers (PLNE) ou de programmation par contraintes (PPC). LocalSolver est un solveur de type *model & run*, c'est-à-dire qu'il suffit de déclarer un modèle dans un formalisme mathématique simple et la résolution est ensuite totalement automatique. Grâce à une recherche locale pure et directe, LocalSolver est capable de trouver très rapidement des solutions de grande qualité pour des problèmes de très grandes tailles.

A première vue on pourrait considérer ici que les décisions à prendre sont les affectations de chaque opération à un pas de temps. En fait une approche plus simple consiste à se focaliser uniquement sur la configuration de la machine a chaque pas de temps c'est-à-dire à la couleur peinte à chaque pas de temps. L'exécution des travaux découle alors directement de cette séquence

de couleurs. Partant de ces variables de décisions décrivant la couleur de chaque pas de temps, on introduit en effet des variables *progression[j][k]* représentant l'avancement du travail *j* au pas de temps *k* c'est-à-dire égales au numéro de la prochaine opération à réaliser pour ce travail quand commence le pas de temps *k*. La couleur sélectionnée pour le pas de temps *k* détermine alors l'avancement en résultant au pas de temps *k+1*, selon que cette couleur correspond ou non à la couleur attendue par ce travail pour sa prochaine opération. Ces enchaînements logiques s'écrivent très simplement en LocalSolver grâce à la disponibilité d'opérateurs fortement non linéaires comme les conditions logiques (If-Then-Else) ou les indexeurs de tableau ($x \leftarrow A[y]$).

```
for[j in 1..NumJob][k in 1..TotalNum] {
    if (k == 1) progress[j][k] <- 1;
    else progress[j][k] <- match[j][k-1] ? progress[j][k-1]+1 : progress[j][k-1];
    expectedColor[j][k] <- Operation[j][progress[j][k]];
    match[j][k] <- expectedColor[j][k] == paintingColor[k];
}
```

Finalement le but est de minimiser le nombre de pas de temps actifs:

```
someMatch[k in 1..TotalNum] <- or[j in 1..NumJob](match[j][k]);
obj <- sum[k in 1..TotalNum](someMatch[k]);
minimize obj;
```

3 Résultats

En pratique les instances rencontrées par *0-1 integer* comportent de 100 à 600 opérations. Les solveurs de PLNE étant impuissants face à des instances de cette taille, c'est une solution à base de programmation par contraintes qui avait été mise en place avec le solveur SCOP [2]. On observe que LocalSolver parvient ici à obtenir des solutions de meilleure qualité avec des temps de calcul bien plus courts.

	Best known		**LocalSolver 10 seconds**	**LocalSolver 60 seconds**	**LocalSolver 600 seconds**
	Value	Time (seconds)			
Data1	49	20s	41	40	40
Data2	87	230s	78	77	75
Data3	125	350s	116	111	108
Data4	193	1250s	174	166	162
Data5	236	2380s	230	217	207

Références

[1] T. Benoist, B. Estellon, F. Gardi, R. Megel and K. Nouioua, LocalSolver 1.x: a black-box local-search solver for 0-1 programming. *4OR, A Quarterly Journal of Operations Research* 9(3), pp. 299-316. Springer, 2011.

[2] K. Nonobe and T. Ibaraki, An improved tabu Search method for the weighted constraint satisfaction problem. INFOR, 39, pp 131-151, 2001.