Modélisation de problèmes d'ordonnancement avec LocalSolver

Léa Blaise[1]

LocalSolver, France
lblaise@localsolver.com

Mots-clés : *ordonnancement, solveur, modélisation.*

1 Introduction

LocalSolver est un solveur d'optimisation mathématique basé sur différentes techniques de recherche opérationnelle, combinant des méthodes exactes, telles que la programmation linéaire, non linéaire et par contraintes, et heuristiques, comme la recherche locale[2]. Son but est d'offrir une approche de type « model-and-run » à des problèmes d'optimisation (combinatoires, continus, mixtes...), y compris sur de grandes instances.

On cherche ici à montrer comment le formalisme ensembliste très riche de LocalSolver permet de modéliser efficacement de nombreux problèmes d'ordonnancement industriels, en n'utilisant que des opérateurs génériques. Plus précisément, on se concentrera d'une part sur les avantages apportés par l'utilisation combinée de variables de décision entières et de listes [1] pour modéliser les problèmes d'ordonnancement disjonctif, et d'autre part sur les opérateurs variadiques permettant de représenter en intention et non en extension certaines contraintes globales propres à l'ordonnancement cumulatif.

2 Ressources disjonctives

La présence de ressources disjonctives dans un problème d'ordonnancement se caractérise par des contraintes de non-chevauchement des tâches affectées à une même ressource. Une façon simple d'écrire cette contrainte consiste à exploiter l'ordre des tâches : chaque tâche ne peut commencer qu'après la fin de la tâche précédente.

L'écriture de cette contrainte dans le formalisme de modélisation de LocalSolver se fait en utilisant deux types de variables de décision. D'une part, chaque ressource est représentée à l'aide d'une variable de liste : les éléments de la liste correspondent aux tâches exécutées sur cette ressource, rangées par ordre croissant des dates d'exécution. D'autre part, des variables de décision entières permettent de représenter les dates de début de ces tâches.

```
order <- list(nbTasks);
start[0..nbTasks-1] <- int(0, horizon);
```

On écrit alors la contrainte de non-chevauchement à l'aide d'un opérateur « et » variadique. Cette contrainte se lit « pour toute position i dans la liste, la tâche en position $i+1$ doit commencer après la date de fin de la tâche en position i ». Cette formulation à base de variables de listes a l'avantage de permettre à l'utilisateur d'écrire la contrainte de non-chevauchement en $O(n)$ (avec n le nombre de tâches) au lieu de $O(n^2)$ en n'utilisant que des variables entières.

```
constraint and(0..nbTasks-2, i => start[order[i+1]] >= start[order[i]] + duration[order[i]]);
```

1. Dans le formalisme de modélisation de LocalSolver, une variable de liste de domaine n est une variable de décision dont la valeur est une permutation d'un sous-ensemble de $\{0, ..., n-1\}$. Les listes sont fréquemment utilisées pour représenter des ordres : ordre des tâches sur une machine en ordonnancement, ordre des points à visiter dans les problèmes de tournées de véhicules, etc.

Ces contraintes sont à la base de nombreux problèmes d'ordonnancement disjonctif, comme par exemple celui du Job Shop[3].

Lorsque le choix de la ressource exécutant chacune des tâches fait également partie des décisions du problème (flexibilité), on utilise une variable de liste par ressource disjonctive, et on contraint ces listes à former une partition de l'ensemble des tâches.

```
order[0..nbResources-1] <- list(totalNbTasks);
constraint partition[m in 0..nbResources-1](order[m]);
```

3 Ressources cumulatives

Contrairement aux ressources disjonctives, une ressource cumulative peut traiter plusieurs tâches à la fois, dans la limite de sa capacité. Les tâches affectées à une ressource cumulative n'étant alors pas ordonnées, celles-ci se modélisent à partir de variables de décision entières uniquement (dates de début des tâches), et ne nécessitent pas l'utilisation de variables de listes.

La contrainte associée à la présence d'une ressource cumulative peut s'exprimer de la façon suivante. A chaque instant t, la somme des poids des tâches en cours d'exécution sur la ressource doit être inférieure à la capacité de la ressource. Dans le formalisme de modélisation de LocalSolver, la dimension temporelle de cette contrainte peut être exprimée en intention à l'aide d'un opérateur « et » variadique. Grâce à cet opérateur variadique, la contrainte n'est pas « déroulée » de façon extensive sur tout l'horizon temporel.

```
constraint and(0..horizon, d => sum[t in tasks](weight[t] * (start[t] <= d && end[t] > d)));
```

Cette écriture de la contrainte a deux avantages : elle permet à la fois une modélisation compacte et une évaluation efficace (la contrainte est évaluée uniquement aux points de rupture de la fonction – dates de début des tâches – et non à chaque instant).

Ces contraintes sont à la base de nombreux problèmes d'ordonnancement cumulatif, comme par exemple le Resource-Constrained Project Scheduling Problem[1].

4 Autres contraintes propres à l'ordonnancement

En plus des contraintes caractérisant les ressources disjonctives et cumulatives, le formalisme de LocalSolver permet de modéliser efficacement de nombreuses autres contraintes classiques souvent rencontrées dans les problèmes d'ordonnancement industriels. On montrera ainsi comment modéliser des contraintes de précédence entre deux tâches, les temps de transition entre deux tâches consécutives, la présence de calendriers (plages d'indisponibilité sur certaines ressources par exemple), les incompatibilités entre une tâche et une machine ou entre deux tâches...

On montrera également comment les modèles présentés permettent d'obtenir de très bons résultats sur de nombreux problèmes d'ordonnancement avec LocalSolver 11.5 (gap de 2.23% sur le Job Shop, 1.27% sur le Job Shop Flexible, 0.96% sur le Job Shop Flexible avec temps de transition, 1.4% sur le Resource-Constrained Project Scheduling Problem...) [2].

Références

[1] Christian Artigues, Sophie Demassey, and Emmanuel Neron. *Resource-Constrained Project Scheduling : Models, Algorithms, Extensions and Applications.* ISTE/Wiley, 2008.

[2] Frédéric Gardi, Thierry Benoist, Julien Darlay, Bertrand Estellon, and Romain Megel. *Mathematical Programming Solver Based on Local Search.* John Wiley & Sons, Ltd, 2014.

[3] Eric Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA journal on Computing*, 6(2) :108–117, 1994.

2. Ecart à la meilleure solution connue, en 60s de calcul