Modélisation de ressources disjonctives avec LocalSolver

Léa Blaise[1,2] Thierry Benoist[2] Christian Artigues[1]

[1] LAAS-CNRS, Université de Toulouse, CNRS, INP, Toulouse, France
[2] LocalSolver, 24 Avenue Hoche, Paris, France
lblaise@localsolver.com

Mots-clés : *ordonnancement disjonctif, recherche locale, solveur, modélisation*

1 Introduction et contexte

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 [1]. 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 les différents types de variables de décision offerts par LocalSolver permettent de modéliser efficacement de nombreux problèmes d'ordonnancement disjonctif, en n'utilisant que des opérateurs génériques. Plus précisément, on se concentrera sur les avantages apportés par l'utilisation combinée de variables de décision entières et de listes, représentant respectivement les dates de début des tâches, et leur ordre sur les machines.

2 Modélisation des contraintes de non-chevauchement

La présence de ressources disjonctives dans un problème se caractérise par des contraintes de non-chevauchement des tâches affectées à une même ressource. Ces contraintes peuvent s'écrire de différentes manières, la plus simple et la plus pratique étant d'exploiter l'ordre des tâches : chaque tâche ne peut commencer qu'après la date de fin de la tâche précédente.

Dans le formalisme de LocalSolver, cette formulation implique d'utiliser deux types de variables de décision : des entiers, représentant les dates de début des tâches, et des listes [1], représentant leur ordre. En LSP, le langage de modélisation de LocalSolver, une contrainte de non-chevauchement s'écrit donc avec un "et" variadique sur l'ensemble des tâches :

```
constraint and(1..nbTasks-1, i => start[order[i]] >= end[order[i-1]]);  (1)
```

où `order` est une variable de liste, `start` un tableau de variables entières, et `end` un tableau d'expressions entières (dates de début plus durées fixes ou variables). Cette contrainte variadique se lit "pour tout `i` entre 1 et `nbTasks-1`, `start[order[i]]` est supérieur ou égal à `end[order[i-1]]`".

Ces contraintes sont à la base de nombreux problèmes d'ordonnancement, qu'il s'agisse d'ordonnancement d'atelier (par exemple, le problème du Job Shop) ou de production (par exemple, le problème du Unit Commitment).

Problème du Job Shop. Le problème comporte n jobs, divisés en m activités à réaliser sur chacune des m machines disjonctives. On connaît la durée et la machine attribuée à chaque activité. Les variables de décision sont les dates de début des tâches (ainsi que leur ordre). On peut alors modéliser les contraintes de non-chevauchement des activités réalisées sur une même machine avec m contraintes de taille n, définies par la forme de (1).

1. Dans LocalSolver, une liste de domaine n est une variable de décision dont la valeur est une permutation d'un sous-ensemble de $[0, n-1]$. En ordonnancement, si chaque tâche du problème est associée à un indice entre 0 et $n-1$, la valeur de la liste permet de modéliser l'ordre des tâches.

Une modélisation alternative de ces contraintes peut se faire au moyen de disjonctions : pour chaque paire de tâches t_1 et t_2 affectées à une même ressource, t_2 commence après la fin de t_1, ou inversement. On a alors $O(mn^2)$ contraintes. La formulation précédente a donc l'avantage d'être beaucoup plus compacte.

Problème du Unit Commitment. Dans ce problème, on dispose de plusieurs centrales électriques, et on doit décider des plages sur lesquelles elles seront allumées ou éteintes, en fonction de la demande, mais également de contraintes structurelles : durées minimale et maximale d'une plage de fonctionnement, durée minimale d'une plage de non-fonctionnement. On peut représenter ces plages de fonctionnement comme des tâches optionnelles. Des variables entières modélisent les dates de début et durées de ces tâches, et des variables de listes, représentant l'ordre des tâches sur chaque ressource, permettent de modéliser les contraintes de non-chevauchement de ces tâches selon (1).

Une formulation alternative classique consiste caractériser l'état de chaque usine à chaque instant par des variables booléennes. La formulation à base de tâches est cependant bien plus pratique pour la modélisation des contraintes (durées minimales et maximales des plages de fonctionnement ou non-fonctionnement).

3 Mouvements de recherche locale

En plus d'offrir une modélisation pratique des contraintes de non-chevauchement, l'expression (1) permet de donner une structure forte au modèle, exploitable par le solveur, notamment dans les mouvements de la recherche locale.

Différents mouvements d'ordonnancement sont ainsi implémentés au sein de LocalSolver, en fonction des caractéristiques des ressources et des tâches (durée fixe ou variable par exemple). On peut citer les exemples suivants : insertion d'une nouvelle tâche sur une machine, fusion de deux tâches voisines, changement de ressource, échange de ressource entre deux tâches, échange des dates de début de k tâches affectées à une même ressource, fractionnement d'une tâche, inversion de l'état de fonctionnement de k ressources sur une petite fenêtre de temps...

Ces mouvements exploitent bien l'intéraction entre les variables représentant les dates de début des tâches et leur ordre sur les machines. Par exemple, pour le mouvement de changement de ressource, on retire l'indice de la tâche choisie de la liste de sa machine initiale pour l'ajouter à la liste d'une autre machine à la bonne position, s'assurant ainsi que les tâches sont triées par dates de début croissantes.

4 Résultats

La formulation à base de tâches ainsi que l'ajout de mouvements de recherche locale dédiés permettent d'obtenir de meilleurs résultats sur le problème du Unit Commitment qu'avec un modèle booléen : les solutions sont en moyenne 50% meilleures en dix secondes, et 30% meilleures en deux minutes sur les instances de [2] (entre 10 et 150 usines).

On observe également un gain sur le problème du Job Shop : la formulation compacte des contraintes de non-chevauchement permet d'obtenir un gap moyen de 2.5% en une minute sur les instances classiques des classes FT, LA et ORB, contre 4% avec une modélisation à base de disjonctions.

Références

[1] F. Gardi, T. Benoist, J. Darlay, B. Estellon, and R. Megel. *Mathematical Programming Solver Based on Local Search*, Wiley, 2014.

[2] A. Angulo, D. Espinoza and R. Palma. *Thermal Unit Commitment Instances for Paper : A Polyhedral-Based Approach Applied to Quadratic Cost Curves in the Unit Commitment Problem* `http://www.dii.uchile.cl/~daespino/UC_instances _archivos/portada.htm`