Modélisation des tâches optionnelles dans les problèmes d'ordonnancement avec Hexaly

Léa Blaise

Hexaly, France
lblaise@hexaly.com

Mots-clés : *ordonnancement, solveur, modélisation, préemption, multi-alternatives*

1 Introduction

Hexaly est un solveur global de type *model & run* basé sur des techniques exactes et heuristiques [1]. On s'intéresse ici à ses performances sur les problèmes d'ordonnancement de façon générale (problèmes présentant des contraintes de précédence et ressources disjonctives ou cumulatives, minimisant le makespan, la somme pondérée des retards, etc.). Le formalisme de modélisation d'Hexaly permet d'exprimer facilement de nombreux problèmes d'ordonnancement académiques et industriels, en n'utilisant que des opérateurs génériques. Ces modèles, construits à partir de variables d'intervalles et de listes, ont l'avantage d'être compacts, et permettent ainsi au solveur de traiter des problèmes de grande taille. En effet, les algorithmes implémentés au sein d'Hexaly exploitent cette modélisation et lui permettent d'obtenir de très bonnes performances sur divers problèmes d'ordonnancement : moins de 2% d'écart à l'optimum en une minute et jusqu'à 2000 tâches sur des problèmes classiques comme le Job Shop, le Flexible Job Shop, l'Open Shop, ou le Resource-Constrained Project Scheduling Problem.

Certains problèmes d'ordonnancement restent cependant difficiles à modéliser avec ces seules variables. On montrera ainsi comment l'introduction de variables d'intervalles optionnels dans le formalisme de modélisation d'Hexaly permet d'exprimer facilement certains aspects complexes des problèmes d'ordonnancement, comme la préemption ou les alternatives multiples. On montrera également que les modèles écrits dans le formalisme ensembliste d'Hexaly sont plus compacts que leurs équivalents dans les formalismes de programmation linéaire ou par contraintes utilisés par d'autres solveurs classiques, comme Gurobi ou CP Optimizer.

2 Intervalles optionnels et opérateur *presence*

Un intervalle optionnel est une variable de décision dont la valeur est soit un intervalle de la forme $[a, b[$, vérifiant $a \leq b$, soit *absent*. Ces variables sont utilisées en ordonnancement pour modéliser les plages d'exécution des tâches. La valeur *absent* signifie alors que la tâche correspondant à cet intervalle n'est pas ordonnancée. Afin de faciliter la manipulation des intervalles optionnels, la bibliothèque d'opérateurs d'Hexaly comprend également un opérateur *presence*, renvoyant *vrai* si l'intervalle est présent et *faux* s'il est absent.

Il est généralement plus avantageux d'utiliser des variables d'intervalles optionnels plutôt que des intervalles classiques couplés à des décisions booléennes représentant la présence de ces intervalles. En effet, l'information de la présence et des dates d'exécution d'une tâche est alors portée par une unique décision, ce qui donne plus d'information au solveur quant à la structure du problème, et lui permet d'appliquer des algorithmes pertinents pour le résoudre. De plus, il est possible d'implémenter des comportements par défaut pour les opérateurs ensemblistes s'appliquant sur des intervalles optionnels, ce qui facilite l'écriture des modèles. Par exemple, l'opérateur *hull* prend en entrée un ensemble d'intervalles et renvoie l'enveloppe des intervalles présents, ou *absent* si aucun de ces intervalles n'est présent.

3 Ordonnancement avec alternatives multiples

On considère un problème d'ordonnancement avec alternatives multiples. Dans ce problème, les tâches sont liées par des contraintes de précédence. Les successeurs d'une tâche donnée constituent différentes alternatives, dont seule une doit être réalisée. Chacune de ces alternatives comprend une ou plusieurs sous-tâches. Si une alternative est sélectionnée, alors chacune de ses sous-tâches doit être réalisée. Grâce aux intervalles optionnels et aux opérateurs *presence* et *hull* proposés par Hexaly, le problème s'exprime de façon simple et compacte :

```
subtask[i in 0...n][k in 0...nbTasks[i]] <- optionalInterval(0, H);
alternative[i in 0...n] <- hull[k in 0...nbTasks[i]](subtask[i][k]);
for [i in 0...n] {
    constraint sum[s in successors[i]](presence(alternative[s])) == presence(alternative[i]);
    for [k in 0...nbTasks[i]] constraint presence(subtask[i][k]) == presence(alternative[i]);
}
```

4 Ordonnancement pseudo-préemptif

Un problème d'ordonnancement préemptif est un problème dans lequel les tâches peuvent être interrompues puis reprises. Dans une modélisation pseudo-préemptive, on définit un nombre maximal p d'interruptions pour chaque tâche, que l'on modélise alors comme un ensemble de $p+1$ sous-tâches optionnelles. Au moins une de ces sous-tâches doit être présente, et la somme des durées des sous-tâches présentes doit être égale à la durée de la tâche. Pour faciliter la modélisation, on suppose de plus que si une sous-tâche d'indice k est présente, alors toutes les sous-tâches d'indice $j \leq k$ sont également présentes. Là encore, les variables de décision d'intervalles optionnels et les nombreux opérateurs non-linéaires d'Hexaly (*presence*, *length*, opérateur ternaire...) permettent de modéliser facilement de tels problèmes :

```
subtask[i in 0...n][k in 0...p+1] <- optionalInterval(0, H);
for [i in 0...n] {
    constraint presence(subtask[i][0]);
    for [k in 1...p+1] constraint presence(subtask[i][k]) <= presence(subtask[i][k-1]);
    constraint sum[k in 0...p+1](presence(subtask[i][k])? length(subtask[i][k]): 0) == d[i];
}
```

On suppose que le problème comprend également m ressources disjonctives flexibles : on doit alors décider de l'affectation des sous-tâches aux différentes machines. Dans le formalisme de modélisation d'Hexaly, on utilise alors des variables de listes (décisions dont la valeur est une permutation d'un sous-ensemble de $\{0, ..., n-1\}$.) représentant l'ordre des tâches affectées aux machines. Le modèle comporte donc $np + m$ décisions au total. En comparaison, un modèle écrit pour un solveur de programmation par contraintes comme CP Optimizer [2] utilise npm intervalles optionnels : un pour chaque sous-tâche et chaque machine, reliés par des contraintes d'alternatives. Dans un formalisme de programmation linéaire, une formulation classique du problème utilise $O((np)^2m)$ décisions booléennes et entières [3]. Grâce à sa modélisation compacte, Hexaly est alors capable de traiter des instances de bien plus grande taille.

Références

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

[2] P. Laborie, J. Rogerie, P. Shaw, and P. Vilim. IBM ILOG CP optimizer for scheduling. *Constraints*, 23 :210–250, 2018.

[3] C. Özgüven, L. Özbakır, and Y. Yavuz. Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling*, 2010.