Hexaly Optimizer 12.5 : modélisation par intervalles et améliorations des performances pour les problèmes d'ordonnancement et de tournées de véhicules

Julien Darlay

Hexaly, 251 Boulevard Pereire, 75017 Paris
jdarlay@hexaly.com

Mots-clés : *Optimisation, tournées de véhicules, ordonnancement*

1 Introduction

Hexaly Optimizer, anciennement LocalSolver, est un solveur global de type *model & run* basé sur des techniques exactes et heuristiques. Son formalisme d'entrée lui permet d'accepter tout modèle utilisant les opérateurs mathématiques usuels (arithmétiques, logiques, relationnels, etc.) avec des variables continues, entières ou ensemblistes. Les techniques exactes permettent d'obtenir des bornes et de prouver l'optimalité. Les heuristiques permettent de trouver rapidement de bonnes solutions et de passer à l'échelle sur les plus grosses instances.

L'exposé présentera les nouveautés de la version 12.5, avec un nouveau formalisme de modélisation pour les problèmes d'ordonnancement ainsi que de nouveaux opérateurs ensemblistes pour mieux modéliser les contraintes d'exclusions dans les très grands modèles. Cette nouvelle version apporte des améliorations de performances sur les problèmes d'ordonnancement et de tournées de véhicules qui seront illustrées sur des benchmarks de la littérature.

2 Modélisation

Les variables ensemblistes `list` permettent de représenter une séquence ordonnée d'entiers, par exemple la liste des tâches à effectuer sur une machine. Pour représenter la date de début et la date de fin d'une tâche, une variable de type `interval` a été ajoutée au formalisme de modélisation. Les opérateurs relationnels $<$ et $>$ ont été étendus aux intervalles avec la sémantique $A < B$, si l'intervalle A se termine avant le début de l'intervalle B. Les expressions `start(`A`)`, `end(`A`)`, `length(`A`)` et `contains(`A`, `t`)` permettent respectivement de représenter la date de début, la date de fin, la longueur de l'intervalle A et l'expression t est inclus dans A. La combinaison de variables `list` et `interval` permet de modéliser efficacement les problèmes d'ordonnancement disjonctifs et cumulatifs.

L'expression ensembliste `distinct` a été introduite afin de retourner la liste des éléments uniques d'un ensemble. Cette expression est particulièrement utile dans les problèmes d'affectation où un conteneur ne peut pas recevoir qu'un seul type de produit. Il suffit alors d'ajouter une contrainte sur la cardinalité de l'ensemble `distinct` des types des objets présents dans le conteneur : `constraint count(distinct(set, i => type[i])) <= 1`.

L'expression ensembliste `intersect` permet de représenter l'intersection de deux ensembles. Il est possible de modéliser de manière compacte des exclusions de paires d'objets dans un même ensemble en contraignant la cardinalité de l'intersection entre l'ensemble et l'ensemble des valeurs interdites : `constraint count(intersect(set, {1, 2, 5})) <= 1`.

L'expression ensembliste `sort` permet de trier un ensemble selon un critère. Elle est utile en optimisation stochastique lorsque l'incertitude est modélisée par des scénarios et que l'utilisateur souhaite minimiser la valeur d'un décile particulier.

3 Recherche de solutions

La modélisation ensembliste est particulièrement adaptée à l'écriture de problèmes de tournées de véhicules, une variable `list` modélise naturellement la séquence des clients à visiter pour un camion. Hexaly Optimizer 12.5 apporte des améliorations de performances avec l'ajout de nouveaux voisinages de type *destroy & repair* sur les problèmes de *pickup & delivery* avec fenêtres de temps (PDPTW). Pour ces problèmes le gap moyen sur les instances classiques de la littérature passe de 8% à 2.5% en 60s. Un second voisinage basé sur le principe des chaines d'éjection [1] permet d'étendre les performances obtenues aux variantes de PDPTW avec des contraintes d'interdiction entre véhicules et camions.

Les variables `interval` ont été introduites pour simplifier l'écriture des problèmes d'ordonnancement tout en gardant les performances des versions précédentes (moins de 3% de gap en 60s pour des instances à plusieurs centaines de tâches : *jobshop*, *flexible jobshop*, *RCPSP*, etc.). Les techniques de résolution ont été étendues pour gérer l'ordonnancement pseudo-préemptif ainsi que l'ordonnancement producteur-consommateur.

4 Calcul de bornes

Les problèmes de PDPTW ansi que les variantes hétérogènes des problèmes de tournées de véhicules sont maintenant automatiquement reformulés en programmes linéaires en nombres entiers compacts [2] pour la résolutions à l'optimum des instances à quelques dizaines de clients. Cette reformulation compacte vient s'ajouter aux reformulations par génération de coupes déjà présentes dans le solveur pour le calcul de bornes inférieures.

Les bornes déjà calculées pour les problèmes d'ordonnancement [3] ont été renforcées par la résolution exacte de relaxations combinatoires par des algorithmes de la littérature [4]. L'idée est ici de se ramener à des problèmes suffisamment simples en éliminant des contraintes pour qu'ils puissent être résolus par des algorithmes dédiés efficaces. Cela permet d'améliorer les bornes calculées sur toutes les instances des problèmes où l'objectif est de minimiser la somme pondérée des dates de fin.

Références

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

[2] H. D. Sherali, P. J. Driscoll On Tightening the Relaxations of Miller-Tucker-Zemlin Formulations for Asymmetric Traveling Salesman Problems. *Operations Research 2002 50(4) :656-669.*

[3] P. Laborie. Bornes rapides pour l'ordonnancement dans LocalSolver. *ROADEF 2023.*

[4] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly 1956.*