Modélisation de problèmes d'ordonnancement par batches avec Hexaly

Agathe L'HERMITE, Léa BLAISE

Hexaly, 251 Boulevard Pereire, 75017 Paris

{alhermite, lblaise}@hexaly.com

Mots-clés : *ordonnancement, problèmes de batches*

1 Introduction

Hexaly est un solveur global de type *model & run* basé sur des techniques de modélisation exactes et heuristiques [1]. On s'intéresse ici à ses performances sur les problèmes d'ordonnancement par batches, où les tâches doivent être regroupées en lots afin d'être effectuées. Le formalisme de modélisation d'Hexaly permet d'exprimer facilement les problèmes de batches. Cette modélisation a l'avantage d'être compacte, notamment grâce à l'utilisation de variables ensemblistes.

Nous présentons ici le Single Machine Total Weighted Tardiness Scheduling Problem (TWTBSP). Nous montrerons comment modéliser les problèmes de batches avec Hexaly. Nous exposerons également comment le mécanisme de recherche locale a été adapté aux problèmes d'ordonnancement par batches. Ces améliorations ont permis de diminuer l'écart à la meilleure solution de 50% à 20% en moyenne sur des instances de 10 à 100 tâches en 30 secondes.

2 Formulation d'un problème d'ordonnancement par batches

Le problème TWTBSP comporte une seule machine, sur laquelle des tâches doivent être regroupées en lots, chaque tâche ayant une taille et les lots ayant une capacité maximale. La durée d'un batch doit être supérieure au maximum des durées des tâches qu'il contient. L'objectif est la minimisation de la somme pondérée des retards, chaque tâche possédant une date d'échéance ainsi qu'une date de disponibilité. Dans Hexaly, les variables d'intervalles permettent de représenter les plages d'exécution des tâches. Nous utilisons dans ce modèle une variable d'intervalle par batch :

```
batchInterval[0...nbBatches] <- interval(0, timeHorizon);
```

Nous formulons la contrainte de disjonction de ressource :

```
for [b in 0...nbBatches]{
    constraint batchInterval[b-1] < batchInterval[b];
}
```

Hexaly inclut également des variables ensemblistes appelées `set`. Une variable `set(n)` décrit un sous-ensemble de $\{0, \ldots, n-1\}$ et représente ici le contenu d'un batch.

```
batchContent[b in 0...nbBatches] <- set(nbTasks);
```

Chaque tâche doit être présente dans exactement un set et nous pouvons exprimer cette contrainte grâce à l'opérateur `partition` appliqué à une famille de sets :

```
constraint partition(batchContent);
```

La somme des tailles des tâches dans un batch ne peut pas dépasser la capacité du batch :

```
for [b in 0...nbBatches]{
    constraint sum(batchContent[b], t => size[t]) <= capacity;
}
```

La longueur de chaque batch doit être supérieure au maximum des longueurs de ses tâches et un batch doit débuter après le maximum des dates de disponiblités de ses tâches. Nous utilisons l'opérateur `find` pour retrouver le batch correspondant à chaque tâche.

```
for [t in 0...nbTasks]{
    chosenBatch[t] <- find(batchContent, t);
    constraint length(batchInterval[chosenBatch[t]]) >= duration[t];
    constraint start(batchInterval[chosenBatch[t]]) >= release[t];
}
```

Enfin, l'objectif est la somme pondérée des retards :

```
TWT <- sum(0...nbTasks, t => weight[t] * max(0, end(batchInterval[chosenBatch[t]]) -
    dueDate[t]));
minimize TWT;
```

3 Agrégation de mouvements et réparation des solutions

Des heuristiques de recherche locale ont été développées dans la littérature pour les problèmes de batches et sont prometteuses [2]. Nous avons donc cherché à améliorer les performances de la recherche locale du solveur. Des mouvements d'intervalles et sets sont intégrés à la composante recherche locale d'Hexaly, avec activation automatique selon les caractéristiques du problème. Les mouvements de sets sont activés sur les problèmes de batches mais ils ne sont pas très performants sur ces problèmes parce qu'ils ne modifient que le contenu des batches et pas les longueurs des intervalles associés. Nous avons réutilisé les mouvements de sets existants en les couplant à une modification des longueurs des intervalles associés aux sets modifiés. En effet, un mouvement modifiant les contenus de sets devrait aussi modifier la longueur des intervalles associés. Nous présentons dans l'algorithme 1 le principe des mouvements dédiés aux problèmes de batches.

Algorithme 1 Adaptation des mouvements aux problèmes de batches

Requiert : Un mouvement modifiant un ou plusieurs sets
  Appliquer le mouvement
  Pour chaque set modifié, **faire** :
    Retrouver l'intervalle associé
    Calculer et mettre à jour la nouvelle longueur de l'intervalle
  Fin pour

Ces nouveaux mouvements permettent d'atteindre des voisinages jusqu'alors inacessibles pour le solveur et de diminuer l'écart relatif aux meilleures solutions de 50% à 20% en 30 secondes sur les instances présentées dans [2] possédant entre 50 et 100 tâches.

Références

[1] Frédéric Gardi, Thierry Benoist, Julien Darlay, Bertrand Estellon, and Romain Megel. *Mathematical programming solver based on local search.* John Wiley & Sons, 2014.

[2] Eduardo Queiroga, Rian GS Pinheiro, Quentin Christ, Anand Subramanian, and Artur A Pessoa. Iterated local search for single machine total weighted tardiness batch scheduling. *Journal of Heuristics*, 27(3) :353–438, 2021.