Génération aléatoire de modèles pour tester la robustesse du solveur d'optimisation Hexaly Optimizer

Ulysse Prieto, Lucas Langlade

Hexaly, 251 Boulevard Pereire, Paris, France
uprieto@hexaly.com
lucas.langlade@eleves.enpc.fr

Mots-clés : *Optimisation, génération de modèles, robustesse, ordonnancement*

1 Introduction

Hexaly Optimizer, anciennement LocalSolver [1], 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.

Hexaly Optimizer est un solveur d'optimisation générique et se doit de pouvoir trouver des solutions de bonne qualité sur des problèmes variés, souvent très éloignés de ceux présents dans la littérature. Pour tester le bon fonctionnement et la qualité des solutions produites, le solveur est régulièrement testé sur un grand nombre d'instances académiques, mais aussi tirées de cas clients.

Hexaly Optimizer 12.0 a introduit de nombreux nouveaux opérateurs de modélisation et notamment un nouveau type de variables de décision : les variables de type `interval`. Le manque d'instances industrielles utilisant ces fonctionnalités peut rendre complexe la vérification de leur bon fonctionnement, en particulier sur des modèles éloignés des instances académiques.

L'exposé présentera un retour d'expérience sur une stratégie utilisée au sein d'Hexaly pour générer des modèles et instances variées afin de pouvoir vérifier que le solveur se comporte de la manière attendue.

2 Génération de modèles aléatoires

L'écriture d'un modèle Hexaly est basée sur la construction d'un graphe acyclique d'évaluation, un exemple est disponible dans la FIG. 1. Une première méthode pour générer des modèles aléatoires est de construire un graphe d'évaluation de manière incrémentale. Pour cela, on commence par définir un ensemble d'expressions E composé initialement de variables de décisions et de constantes. Cet ensemble est ensuite étendu de manière incrémentale par sélection uniforme d'une ou plusieurs expressions dans E et d'un opérateur de modélisation adapté à leur type. Les objectifs et contraintes du modèle sont alors sélectionnés parmi les expressions ayant un type adapté, respectivement scalaire et booléen.

Cette première méthode de génération a l'avantage d'être très simple à mettre en œuvre mais ne génère quasiment jamais de structure dans le modèle. Le but de la génération est de tester la robustesse du solveur sur des problèmes d'ordonnancement. Le générateur a ainsi été modifié pour considérablement augmenter la probabilité de créer des structures spécifiques à ce type de problèmes, en particulier des contraintes de précédences, de ressources disjonctives et cumulatives (voir FIG. 2).

```
task <- interval(0, 100);
obj <- 0.1 * start(task);
```

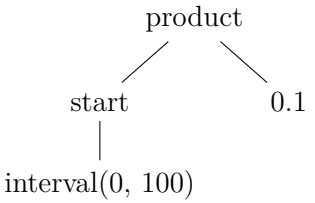

FIG. 1 – Exemple de graphe d'évaluation d'un modèle Hexaly

```
constraint task1 < task2;
constraint and(1...count(order), i => tasks[order[i-1]] < tasks[order[i]]);
constraint and(0...H, t => sum[task in tasks](contains(task, t)) <= Capacity);
```

FIG. 2 – Structure de contraintes de précédences, de ressources disjonctives et cumulatives

3 Génération de modèles admettant une solution faisable

La méthode de génération présentée ci-dessus permet déjà de détecter des modèles sur lesquels le solveur ne se comporte pas de la manière attendue. Cependant, cette méthode possède l'inconvénient de ne générer que très peu de modèles faisables. Nous l'avons donc modifiée afin de s'assurer qu'elle crée des modèles admettant au moins une solution faisable. L'idée est d'assigner au début de la génération une valeur à chaque variable de décision et de maintenir la valeur de l'ensemble des expressions du modèle. Les contraintes et structures spécifiques à l'ordonnancement sont alors générées de manière à respecter l'ensemble des valeurs des décisions.

Savoir que le modèle généré possède une solution faisable est très utile pour vérifier le bon fonctionnement du solveur. On peut par exemple tester si le solveur trouve ou non une solution faisable et ainsi détecter des instances problématiques.

4 Réduction de modèles

L'étude des instances problématiques trouvées par génération aléatoire peut s'avérer complexe, surtout si les modèles et instances sont de grande taille. Pour simplifier ce processus, nous avons implémenté une méthode de réduction basée sur l'algorithme ADEL [2] de recherche de conflit minimal. Cet algorithme va chercher à sélectionner un sous-ensemble minimal des contraintes et objectifs d'un modèle problématique de telle sorte à ce que le sous-modèle engendré par les expressions sélectionnées continue à poser souci. Cette méthode permet en pratique de très fortement réduire la taille d'une majorité des instances problématiques et ainsi de simplifier leur analyse.

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] Philippe Laborie. An optimal iterative algorithm for extracting mucs in a black-box constraint network. In *ECAI 2014*, pages 1051–1052. IOS Press, 2014.