INFO-F-524 : Optimisation Continue
Projet

Bernard Fortz Marcio Costa Santos Jérôme De Boeck

Mars 2017

1 Problème

Le problème de bin packing est un problème combinatoire intéressant avec des applications industrielles, dans lequel un ensemble de produits P doit être emballé dans un ensemble de boîtes B. Chaque produit a une taille $s_p > 0$ et chaque boîte a une capacité de $c > 0$, toutes les boîtes ont la même capacité. Le but est de trouver un rangement qui utilise le moins de boîtes possibles.

2 Formulations

2.1 Formulation 1

En utilisant les variables de décisions suivantes :
— $u_b \in \{0, 1\}$, $u_b = 1 \Leftrightarrow$ on utilise la boîte b,
— $x_{pb} \in \{0, 1\}$, $x_{pb} = 1 \Leftrightarrow$ on dépose le produit p dans la boîte b,
on peut formuler le problème de bin packing comme un programme linéaire en variables entières de la manière suivante.

$$\begin{aligned}
&\min_x \sum_{b=1}^{n} u_b \\
&\text{s.t} \sum_{p=1}^{n} s_p x_{pb} \leq u_b c && \forall b \in \{1, \ldots, n\} \\
&\sum_{b=1}^{n} x_{pb} = 1 && \forall p \in \{1, \ldots, n\}
\end{aligned}$$

2.2 Formulation 2

Une autre manière de formuler le problème est d'utiliser des variables représentant les packings possibles. Nous supposons que les objets sont indexés de 1 à n.

— Soit $\mathcal{S}$ l'ensemble des sous-ensembles d'objets de $\{1, \ldots, n\}$ pouvant rentrer dans une boîte. Aucun ensemble d'objets $S \in \mathcal{S}$ ne dépasse la capacité d'une boite. Pour un ensemble $S \in \mathcal{S}$, la variable $z_S \in \{0, 1\}$ indique si une boîte est utilisée pour mettre les objets de S sans aucun autre objet supplémentaire.

Avec ces variables, on peut formuler le problème comme suit :

$$\begin{aligned} &\min_x \sum_{S \in \mathcal{S}} z_S \\ &s.t \sum_{S \in \mathcal{S}, p \in S} z_S = 1 \qquad\qquad \forall p \in \{1, \ldots, n\} \end{aligned}$$

3 Description du projet

Le projet porte sur l'utilisation de techniques d'optimisation pour obtenir rapidement des bornes inférieures et supérieures sur la valeur optimale du problème. Les implémentations doivent être faites en Python3 et/ou en Xpress. Une version étudiante de Xpress est disponible sur le site officiel. Votre code doit impérativement fonctionner sur les ordinateurs du bâtiment NO. La documentation pour Xpress peut être trouvée ici [1]. Nous vous proposons un projet composé des 5 tâches suivantes.

Tâche 1 *Résoudre la relaxation linéaire de la Formulation 1.*

Tâche 2 *Développer une heuristique qui trouve une solution entière admissible. L'heuristique peut être implementée avec un des langages proposés.*

Tâche 3 *Résoudre une relaxation lagrangienne de la Formulation 1* ***ou*** *résoudre la Formulation 2 par génération de colonnes.*

Relaxation Lagrangienne

1. *Justifiez le choix des contraintes relachées.*
2. *Expliquez dans le rapport comment le sous-problème à chaque itération de la méthode est résolu.*

Consigne 1 *Il faut avoir en tête que la relaxation Lagrangienne à deux avantages par rapport à relaxation linéaire ordinaire, la plus petite taille des sous-problèmes et la possibilité de décomposer ces sous-problèmes. Un choix judicieux des contraintes à relacher permet d'exploiter ces deux aspects ce qui est attendu si vous choisissez cette méthode de résolution.*

Génération de colonnes

1. *Expliquez dans le rapport comment le sous-problème à chaque itération de la méthode est résolu.*

Consigne 2 *La génération de colonnes est efficace si les solutions de bases de départ sont bonnes. Il sera donc attendu qu'une bonne solution de départ soient trouvée si vous utilisez cette méthode pour résoudre la Formulation 2.*

Que vous utilisiez la relaxation Lagrangienne ou la génération de colonne relâchée, veillez à exploiter au maximum les avantages des différentes méthodes.

Tâche 4 *Développer une heuristique basée sur la solution obtenue par la relaxation de la tâche précédente.*

1. *Expliquez dans le rapport comment la solution de la relaxation est modifiée.*

Consigne 3 *Les multiplicateurs lagrangien sont une source d'informations et peuvent être exploités.*

Consigne 4 *Bien comprendre le sous-problème qui doit être résolu a chaque itération de la génération de colonnes est aussi une manière de développer une heuristique.*

Tâche 5 *Comparer les meilleures solutions obtenues (optimum et temps de résolution) avec les heuristiques, par rapport à celle trouvées avec la relaxation choisie.*

Tâche 6 *Comparer l'influence du nombre d'objets de l'instance et la taille des boites sur les temps de résolution.*

Certaines références pourront vous être utiles en plus des descriptions des méthodes présentes dans le cours théorique : [3], [2].

4 Informations Pratiques

Votre code sera testé sur les machines du NO, veillez à indiquer dans votre rapport les commandes à exécuter pour résoudre les problèmes d'optimisation des différentes tâches.

Des instances sont disponibles sur l'UV pour effectué vos tests. Ces instances proviennent de l'université de Bologne et sont tirées de la librairie *Falkenauer* à la page :
`http://or.dei.unibo.it/library/bpplib`.
Une description des instances est disponible à cette page sous le titre *Benchmarks*. Deux jeux d'instances vous sont proposées, un jeu avec des boîtes de taille 150, un autre avec des boites de taille 1000.

Votre rapport doit être composé d'une description de vos implémentations et des résultats de vos tests ainsi que les descriptions demandées pour les tâches 2,3 et 4. Il faut enregistrer pour chaque instance les solutions trouvées, leurs objectif ainsi que tout autre paramètre que vous trouvez pertinent. Le temps d'exécution peut être limité à 30 minutes pour vos différents algorithmes.

Les codes sources et le rapport final doivent être envoyés à bfortz@ulb.ac.be pour le 26 mai 2017 à 12h30.

Références

[1] DashOptimization. Mosel user guide, 2008. http://homepages.ulb.ac.be/~bfortz/moselug.pdf.

[2] Francois Vanderbeck and Laurence Wolsey. Reformulation and decomposition of integer programs. *CORE Discussion paper*, 16, 2009. http://www.uclouvain.be/cps/ucl/doc/core/documents/coredp2009_16.pdf.

[3] Laurence A. Wolsey. *Integer Programming*. Wiley Interscience Publications, New York, 1998.