Benders Decomposition

Alexandre Do, Paul-Alexis Dray, Kamil Rachidi and Jean-François Thai

January 25, 2018

Abstract

Dans ce rapport, nous nous proposons de mettre en oeuvre la décomposition de Benders, pour résoudre des problèmes de la forme :

$$\begin{aligned} \text{minimize} \quad & c^T x + f^T y \\ \text{subject to} \quad & Ax + By \leq b, \\ & x, y \geq 0 \end{aligned} \tag{1}$$

1 Introduction

Nous avons réalisé notre implémentation en **C++**, en utilisant le solveur **Clp** du projet Coin-or pour la résolution de problèmes d'optimisation.
Dans la suite, nous introduirons les principales classes de la librairie Clp que nous avons utilisées, avant de présenter en détail l'algorithme que nous avons implémenté. Pour finir, nous discuterons des résultats de nos expérimentations.

2 La librarie Clp

Clp stands for: ***C****oin-or* ***L****inear* ***P****rogramming.*
Clp est un solveur de problème linéaire programmable open-source, écrit en C ++. Il fait partie du projet Coin-Or, et est conçu pour trouver des solutions de problèmes d'optimisation mathématique de la forme :

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & row_{lb} \leq Ax \leq row_{ub}, \\ & col_{lb} \leq x \leq col_{ub} \end{aligned} \tag{2}$$

Dans notre implémentation, nous utilisons :

- la classe *OsiClpSolverInterface* qui est le solveur de base : il suffit de renseigner le vecteur de la fonction objective, la matrice des contraintes, le vecteur du second membre et les bornes des variables.
- le format de fichier MPS (Mathematical Programming System) qui permet de représenter des problèmes de programmation linéaire ou d'entiers mixtes.

3 L'algorithme de décomposition de Benders

3.1 Principe

L'algorithme de Benders pour la résolution de problème d'optimisation de la forme 1 s'écrit de la manière suivante :

Algorithm 1 Benders Decomposition

Input: lp un problème de la forme 1
  Initialiser $\hat{y}$, $UB = +\infty$, $LB = -\infty$
  while $UB - LB > \epsilon$ **do**
    $\hat{w} \leftarrow$ solution du sous-problème
    if $\hat{u}$ non borné **then**
      ajouter la coupe de faisabilité $[b - By]^T \hat{u} \leq 0$ au problème principal restreint
    else
      ajouter la coupe d'optimalité $f^T y + [b - By]^T \hat{u} \leq z$ au problème principal restreint
      $LB = f^T y + [b - By]^T \hat{u}$
    end if
    $\hat{z} \leftarrow$ solution du problème principal restreint
    $UB = \hat{z}$
  end while

3.2 Notre implémentation

Pour implémenter l'algorithme 1, nous avons procédé de la manière suivante:

1. nous lisons le problème à partir d'un fichier MPS
2. nous récupérons ses différents paramètres
3. nous mettons toutes les valeurs du vecteur $\hat{y}$ à 5 (choix arbitraire)
4. nous générons le premier sous-problème, grâce à notre fonction $generateSubProblem$
5. nous résolvons le sous-problème et générons le premier problème principal restreint, grâce à notre fonction $generateMasterProblem$

Nous répétons ensuite les étapes 4 et 5, en ajoutant des coupes sur le problème principal restreint grâce à notre fonction $updateMasterProblem$ tant que la diffèrence entre la borne inférieure et supérieure est supérieure à ϵ défini à 0,01

4 Commentaires sur nos résultats

Nous avons testé notre programme sur trois problèmes (fournis dans le dossier dat).

4.1 Problème 1

Le premier problème que nous avons soumis à notre programme est l'exemple présenté en fin du cours 5 et traité en classe :

$$\begin{aligned} \text{minimize} \quad & 2x_1 + 3x_2 + 4y_1 + y_2 \\ \text{subject to} \quad & x_1 + x_2 + y_1 + y_2 \geq 9.5, \\ & x_1 + 2x_2 + y_1 \geq 3.5, \\ & 3x_1 + 2x_2 \geq 1.5, \\ & x_1 + y_1 \geq 0.5, \\ & x_2 \geq 0.5, \\ & x, y \geq 0 \end{aligned} \tag{3}$$

Sur ce premier problème, notre programme se comporte exactement comme nous le souhaitons et suit parfaitement les étapes que nous avons détaillées en classe : nous trouvons le résultat optimal attendu sans aucune difficulté.

4.2 Problème 2

Le deuxième problème que nous avons soumis à notre programme est une variante de l'exemple traité dans le TP2 :

$$\begin{aligned} \text{minimize} \quad & -x_1 - 2y_1 \\ \text{subject to} \quad & -2x_1 + 2y_1 \geq 3, \\ & 2x_1 + 2y_1 \geq 9, \\ & 9x_1 - 4y_1 \geq 21, \\ & x, y \geq 0 \end{aligned} \tag{4}$$

Sur ce problème, notre programme ne parvient pas à trouver la solution optimale. Cela pourrait être dû à un problème dans la génération de coupe de faisabilité.

4.3 MKnapsack

Enfin, pour le dernier problème, nous avons généré aléatoirement un problème de 20 variables soumis à 10 contraintes, en utilisant une variante de la procèdure donnée dans le fichier *partie − a.cpp* du TP2. Nous avons procédé comme suit : on génére le vecteur $[c, f]$, la matrice $[A, B]$, puis le second membre b.

Pour ce problème, notre programme ne semble pas rencontrer de problème : nous trouvons la solution attendue (calculée avec le solveur Clp). Cependant, nous devons noter qu'au cours de ces itérations, nous n'avons pas rencontré de cas où le sous-problème est non borné (cas qui pourrait être difficile à gérer cf. Problème 2).