Objectifs

- ☞ Prise en main de **`Coin-clp`**
- ☞ Prise en main de **`Coin-cbc`**

Exercice 1 — [Gestion des coupes à l'aide de `Coin-Cbc`]

Cet exercice est la suite du dernier TP introduisant l'optimisation à l'aide du solver **coin-cbc**. Son objet est d'illustrer l'ajout de coupes dans un solver **`OsiClp`**. Il est possible d'ajouter une *coupe ligne (row cut)* ou bien une *coupe colonne (column cut)*. Une coupe ligne n'est autre qu'une nouvelle contrainte. Quant à une coupe colonne c'est l'ajout d'une nouvelle variable au problème. Nous traiterons le cas des coupes lignes, l'autre cas se traite de manière similaire.

Comme illustration, considérons le problème non borné suivant :

$$\begin{array}{ll} \mathit{Min} & 3x + y \\ s.c. & \\ & -2x + y \leq 0, \\ & x - 2y \leq 0, \\ & x, y \geq 0. \end{array} \tag{1}$$

L'exemple **`partie-c.cpp`** illustre au moins deux choses. La première est comment récupérer un rayon extrême dans le cas d'un problème non borné. La deuxième, comment ajouter et supprimer une coupe ligne.

1. Analyser le code **`partie-c.cpp`**.
2. Coder à l'aide **`coin-clp`** ou **`coin-cbc`** l'exemple (1) (vous pouvez, réutiliser vos codes du dernier TP). Puis, lui ajouter la coupe :

   $$x + 2y \leq 7,$$

   puis

   $$2x + 3y \leq 11.$$

   A chaque fois, quelles est la nouvelle solution optimale obtenue ? Obtenez-vous les mêmes résultats qu'en TD ?

Problème — [Génération de colonnes avec AMPL]

L'objectif de ce problème est de vous guider dans l'implémentation d'un algorithme de génération de colonnes à l'aide **`d'AMPL`**.

Nous avons déjà abordé le principe de génération de colonnes avec l'algorithme de décomposition de *Dantzig-Wolf*. Ce principe a été utilisé pour pallier l'explosion combinatoire des points extrêmes et rayons extrêmes du modèle utilisé.

Dans ce TP nous utiliserons ce même principe pour résoudre un *problème de découpe industrielle*. qui s'énonce comme suit. Un tisserand dispose de rouleaux de tissu, tous de même largeur standard (sa valeur est sans importance !) et de même longueur L (pour faciliter le problème). Nous supposerons également que seules n longueurs de découpe sont possibles. Chaque commande que reçoit le tisserand indique la longueur souhaitait (parmi les n possibles) ainsi que son nombre. Nous noterons l'ensemble des commandes comme suit $\{(l_i, d_i) : i = 1, \ldots, n\}$.

L'objectif du tisserand est de satisfaire l'ensemble de ses clients tout en minimisant le nombre de rouleaux à utiliser.

Partie A : Modélisation

Une façons de modéliser ce problème est d'introduire la notion de *motif*. Un motif est une manière particulière de découper un rouleau de tissu. Autrement dit, un motif est une liste de longueurs de somme totale au plus L (la longueur de rouleau). Notez que le nombre de motif peut devenir très grand avec le nombre de longueurs de découpe et L !

Notons par K le nombre, inconnu, de motifs possibles. Soit, pour tout entier k appartenant à $\{1, \ldots, K\}$, la variable x_k indiquant le nombre de fois que le motif k est utilisé dans la découpe et notons par v^k le vecteur à n composantes indiquant les longueurs de découpe utilisées dans le motif k. Également, notons par d le vecteur à n composantes indiquant les n commandes.

1. A l'aide des notations ci-dessus modéliser le problème.
2. Quelles conditions que doit satisfaire chaque vecteur $v^k, k \in \{1, \ldots, K\}$?

Partie B : Mécanisme de génération de colonnes

Pour des valeurs de L et n assez grandes il est impossible de résoudre le problème directement. Pour contourner cela, le principe de génération de colonnes sera d'un grand secours. Comment le mettre en œuvre ? C'est le but de cette partie.

Pour mettre en œuvre le mécanisme de sélection des colonnes, il nous faudra, d'une part, relaxé les contraintes d'intégrité sur les variable x_k dans le problème (**??**). Nous disposerons ainsi des multiplicateurs duaux associés à la solution optimale du problème relaxé. D'autre part, il nous faudra généré K_0 colonnes initiales (cf. initialisation).

Notons par μ le vecteur des multiplicateur simplexe associé à la solution optimale du problème relaxé et réduit initial.

1. Écrire le dual de la relaxation continue du problème maitre restreint.
2. Quels sont les coûts réduits associés aux variables x^k ?
3. A quelles conditions sur les coûts réduits reconnaitrait-on la solution optimale du problème maitre restreint ?
4. Proposer un problème d'optimisation permettant de vérifier ces dernières conditions.

Concernant l'initialisation, nous pouvons proposer la solution initiale suivante :

$$v_j^k = \delta_{kj} \frac{L}{l_j}, \forall\, k, j = 1, n, \tag{2}$$

où δ_{kj} est le symbole de Chronecker.

L'initialisation dépend du problème à résoudre !

Partie C : Mise en œuvre avec AMPL

Pour pouvoir mettre en œuvre l'algorithme de génération de colonnes à l'aide **d'AMPL** il est nécessaire de savoir, entre autres, comment gérer les colonnes (leur nombre n'est pas connu à l'avance !) et aussi comment gérer le modèle primal restreint et le modèle auxiliaire. Ces deux point font l'objet de cette dernière partie.

Nous supposerons que les définitions du problème maitre restreint et du sous-problème seront faites dans un fichier **.mod** et que les instructions définissant l'algorithme de génération de colonnes seront données dans un fichier **.run** (un tel fichier n'est autre qu'un script **ampl**).

Voici quelques indications :

☞ Pour définir un ensemble d'indices extensible, il faut en premier définir sa cardinalité comme un paramètre. Puis, définir l'ensemble en question comme celui des indices allant de 1 jusqu'à sa cardinalité.

```
1 param Cardinalite integer > 0;
2 set EnsExtensible := 1 .. Cardinalite;
```

Listing 1 – Définir un ensemble extensible.

☞ Pour définir un *problème* il faut procéder comme indiquer ci-dessous :

```
problem nomProbleme: nomVars,nomObjectif,listeConstraintes;
```

Listing 2 – Définir un problème.

Aussi, il est possible d'associer des options de résolution à chaque problème. Ci-dessous, nous définissons la relaxation continue à partir du modèle entier.

```
problem RelaxContinue: varsEntieres,n Objectif,listeConstraintes;
 option relax_integrality 1;
 option solver nomSolver
```

Listing 3 – Définir un problème.

☞ La boucle suivante sera utile :

```
repeat {
 !...
};
```

Listing 4 – Boucle répéter.

pour sortir d'une telle boucle vous pouvez utiliser l'instruction **`break`**.

A présent voici à vous de compléter les fichiers **`.mod`** et **`.run`** ci-dessous.

```
!---
!--- Fichier .mod :: Parametres utiles ...
param taille_rouleau integer > 0; !--taille d'un rouleau
set LongDEMANDEES;
param DEMANDE {LongDEMANDEES} > 0;
param nbrMotifs integer >= 0;
set MOTIFS := 1 .. nbrMotifs;
param Dmd{LongDEMANDEES,MOTIFS} integer >= 0;
check {j in MOTIFS}:
 sum {i in LongDEMANDEES} i * Dmd[i,j] <= taille_rouleau;
!--
!-- Probleme maitre restreint ..
var ...; # les variables ...
minimize nbrRouleaux:
 ...;
subject to ContDmd {i in LongDEMANDEES}:
 ... ;
!--
!-- Sous probleme ...
param prix {LongDEMANDEES} default 0.0;
var ...;
minimize CoutReduit:
...;
subject to LongLimite:
 ...;
```

Listing 5 – Fichier **`.mod`**

Enfin, voici à quoi peut ressemble votre fichier **`.run`**.

```
!---
!--- Fichier .run
reset;
!-- Options et solvers ...
option solver cplexamp;
!--
model ...;
data ...;
!--
!-- Le pb principal et le sous-pb ...
```

```
problem PbPrincipal: ...;
 option relax_integrality ..;
 option presolve 0;
problem SousPb: ...;
 option relax_integrality 0;
 option presolve 1;
!--
!--L'algorithme
let nbrMotifs := 0;
!-- Solution initiale ...
for {i in LongDEMANDEES} {
 let nbrMotifs := nbrMotifs + 1;
 let Dmd[i,nbrMotifs] := floor (taille_rouleau/i);
 let {i2 in LongDEMANDEES: i2 <> i} Dmd[i2,nbrMotifs] := 0;
};
!--
!-- La boucle ...
repeat {
 !-- resoudre probleme maitre ...
 let {i in LongDEMANDEES} prix[i] := ContDmd[i].dual;
 !-- resoudre sous-probleme
 !-- Ajouter une colonne
 if CoutReduit < 0 then {
  let nbrMotifs := nbrMotifs + 1;
  let {i in LongDEMANDEES} Dmd[i,nbrMotifs] := y[i];
 }else break;
};
!---
!--- Rapport solution ...
printf "\n Rapport solution : \n";
display Dmd;
display x;
printf "\n Resolution exacte du pb principal \n";
option PbPrincipal.relax_integrality 0;
option PbPrincipal.presolve 10;
solve PbPrincipal;
```

Listing 6 – Fichier **.run**