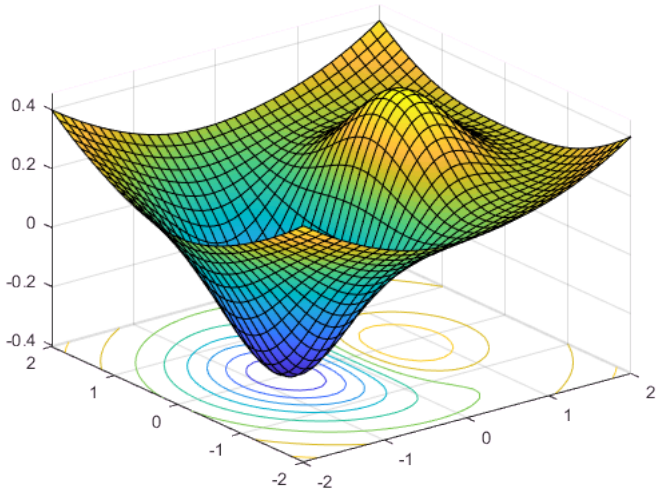
Polytech Sorbonne

Projet de fin de cours d'optimisation linéaire - MAIN4

Problème d'optimisation en variable entière



Aurélien ABEL

Fatine BENTIRES ALJ

Harold GALLICE

Boussad MERHANE

Alexia ZOUNIAS-SIRABELLA

Encadrant :

Hacène OUZIA

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

POLYTECH SORBONNE

Année 2017 - 2018

Table des matières

Première partie

Introduction au problème

Le but de ce projet est de résoudre le problème (P) suivant :

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} (y_{ij} a_{ij} + c_{ij} x_{ij})$$

s.c

$$\sum_{i=1}^{m} x_{ij} \geq A_j, \text{ j} \in \{1, ..., n\}, \tag{46}$$
$$\sum_{j=1}^{n} x_{ij} \leq M_i, \text{ i} \in \{1, ..., m\}, \tag{47}$$

$$m_{ij} y_{ij} \leq x_{ij} \leq M_i y_{ij}, \forall i, j \tag{48}$$
$$y \in \{0,1\}^{m \times n},\ x \geq 0 \tag{49}$$

où :

— $c = (c_{ij}), a = (a_{ij}) et m = (m_{ij})$ sont des matrices appartenant à $M_{m \times n}(\mathbb{R})$,

— $A = (A_j)$ et M $= (M_i)$ sont deux vecteurs appartenant à $\mathbb{R}_m$ et $\mathbb{R}_n$, respectivement

Il s'agit d'un problème d'optimisation en variables entières. Ce problème est **NP-Difficile**, en général.

La *relaxation continue* (cf. introduction) du problème (P) est le problème *d'optimisation linéaire* suivant :

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} (y_{ij} a_{ij} + c_{ij} x_{ij})$$

s.c

$$\sum_{i=1}^{m} x_{ij} \geq A_j, \text{ j} \in \{1,...,\text{n}\},$$
$$\sum_{j=1}^{n} x_{ij} \leq M_i, \text{ i} \in \{1,...,\text{n}\},$$

$$m_{ij} y_{ij} \leq x_{ij} \leq M_i y_{ij}, \forall i, j$$
$$y \in [0,1]^{m \times n},\ x \geq 0$$

La solution optimale de cette relaxation continue n'est, en général, pas entière. En revanche, sa valeur est toujours un minorant de la valeur d'une solution optimale du problème en variables entières.

Considérons le problème (M) équivalent au problème (P) suivant :

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} (y_{ij} a_{ij} + c_{ij} x_{ij})$$

s.c

$$\sum_{i=1}^{m} x_{ij} \geq A_j, \text{ j} \in \{1,...,\text{n}\}, \tag{50}$$
$$\sum_{j=1}^{n} x_{ij} y_{ik} \leq M_i y_{ik}, \forall \text{ i,k} \tag{51}$$
$$\sum_{i=1}^{n} x_{ij} \text{ - } \sum_{i=1}^{n} x_{ij} y_{ik} \leq M_i - M_i y_{ik}, \forall \text{ i,k}, \tag{52}$$

$$m_{ij} y_{ij} \leq x_{ij} \leq M_i y_{ij}, \forall \text{ i,j} \tag{53}$$
$$y \in \{0,1\}^{m \times n}, x \geq 0 \tag{54}$$

Ce modèle (M) s'obtient à partir du modèle (P) en multipliant les contraintes (47) par y_{ik} et 1 - y_{ik} et ce pour toutes les paires d'indices (i, k) que l'intérêt d'un tel modèle est dans sa relaxation continue. Cette manipulation est valide car y_{ik} et 1 - y_{ik} sont positives. Donc, l'ensemble des solutions réalisables du problème (P) n'est pas altéré. En revanche, le modèle (M) ainsi obtenu est plus difficile à résoudre que le modèle originel. Cette difficulté est essentiellement due aux nombres de variables et contraintes du nouveau modèle (M). Nous verrons que l'intérêt d'un tel modèle est dans sa relaxation continue.

Enfin, considérons une linéarisation possible du modèle (M). Cette linéarisation consiste : (i) à remplacer les termes non linéaires $x_{ij}yik$ par de nouvelles variables notées z_{jk}^{i} ; (ii) Puis, à ajouter des contraintes, dites valides, liant les nouvelles variables et les variables originelles. Nous considérerons seulement les contraintes suivantes :

$$z_{jk}^{i} \geq 0, \forall \text{ i,j,k},$$
$$z_{jk}^{i} \leq y_{ij}, \forall \text{ i,j,k}.$$

La validité des deux premiers groupes de contraintes ci-dessus est claire.

Ainsi, le modèle linéarisé (que nous noterons (L)) du modèle (M) est le modèle suivant :

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} (y_{ij} a_{ij} + c_{ij} x_{ij})$$

s.c

$$\sum_{i=1}^{m} x_{ij} \geq A_j, \text{ j} \in \{1,...,\text{n}\}, \tag{55}$$
$$\sum_{j=1}^{n} z_{jk}^{j} \leq M_{ij} y_{ij}, \forall \text{ i,j,k} \tag{56}$$
$$\sum_{i=1}^{n} x_{ij} \text{ - } \sum_{i=1}^{n} x_{ij} y_{ik} \leq M_i - M_i y_{ik}, \forall \text{ i,k}, \tag{57}$$

$$m_{ij} y_{ij} \leq x_{ij} \leq M_i y_{ij}, \forall \text{ i,j}, \tag{58}$$

$$z^i_{jk} \leq x_{ij} \ \forall \text{ i,j,k}, \tag{59}$$
$$y \in \{0,1\}^{m\times n},\ x, z \geq 0. \tag{60}$$

Pour résoudre le problème, nous avons cherché un exemple qui s'y prête. Ainsi, il est possible de visualiser le problème (P) comme un ensemble d'usines de productions.

En effet, nous avons considéré le problème suivant :

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} (y_{ij} a_{ij} + c_{ij} x_{ij})$$

s.c

$$\sum_{i=1}^{m} x_{ij} \geq A_j, \text{ j} \in \{1, ..., n\},$$
$$\sum_{j=1}^{n} x_{ij} \leq M_i, \text{ i} \in \{1, ..., m\},$$

$$m_{ij} y_{ij} \leq x_{ij} \leq M_i y_{ij}, \forall i, j$$
$$y \in \{0,1\}^{m\times n},\ x \geq 0$$

On modélise le problème comme un ensemble de $m \times n$ usines produisant des choses différentes (exemple : chocolat, caramel, etc ...) et on pose :

- x_{ij} : Production en kg de l'$usine_{ij}$ de sa spécialité
- a_{ij} : Coût fixe de l'$usine_{ij}$
- c_{ij} : Bénéfice ($c_{ij} \leq 0$) ou coût (($c_{ij} \geq 0$) de l'$usine_{ij}$ par kg produit
- y_{ij} : Vaut 0 si l'on décide de ne pas ouvrir l'$usine_{ij}$ et 1 sinon
- $max(0, m_{ij})$: Production minimale en kg de l'$usine_{ij}$

Ainsi, les contraintes sur la matrice x s'expliquent. En effet, la somme des éléments d'une ligne représente la production de chaque spécialité qui est majorée par M_i et la somme des éléments d'une colonne est la production des usines par région qui doit être plus grande que A_j pour la satisfaction des clients.

Quatre jeux de données nous sont fournis. Ils seront utilisés dans la suite du projet pour illustrer certaines questions théoriques.

Deuxième partie

Partie A : Résolution exacte

Les premières expériences suggérées dans cette première partie ont pour objectif de vérifier quelques propriétés théoriques liant les deux modèles (P) et (L). Ces expériences seront à effectuer sur quelques instances de petites tailles.

1. **Vérifier l'équivalence des deux modèles (L) et (P).**

Soit le modèle (P) suivant :

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} (y_{ij} a_{ij} + c_{ij} x_{ij})$$

s.c

$$\sum_{i=1}^{m} x_{ij} \geq A_j, \text{ j} \in \{1, ..., n\},$$
$$\sum_{j=1}^{n} x_{ij} \leq M_i, \text{ i} \in \{1, ..., m\},$$

$$m_{ij} y_{ij} \leq x_{ij} \leq M_i y_{ij}, \forall i, j$$
$$y \in \{0,1\}^{m \times n},\ x \geq 0$$

Passons du modèle (P) à sa relaxation continue dans un premier temps.

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} (y_{ij} a_{ij} + c_{ij} x_{ij})$$

s.c

$$\sum_{i=1}^{m} x_{ij} \geq A_j, \text{ j} \in \{1,...,\text{n}\},$$
$$\sum_{j=1}^{n} x_{ij} \leq M_i, \text{ i} \in \{1,...,\text{n}\},$$

$$m_{ij} y_{ij} \leq x_{ij} \leq M_i y_{ij}, \forall i, j$$
$$y \in [0,1]^{m \times n},\ x \geq 0 \Leftarrow \text{y peut prendre plus de valeurs mais est toujours compris entre 0 et 1. Équivalent à } y \in \{0,1\}^{m \times n}.$$

Puis de sa relaxation au modèle (M) dans un second temps.

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} (y_{ij} a_{ij} + c_{ij} x_{ij})$$

s.c

$\sum_{i=1}^{m} x_{ij} \geq A_j$, j $\in$ {1,...,n},
$\sum_{j=1}^{n} x_{ij} y_{ik} \leq M_i y_{ik}$, $\forall$ i,k, $\Leftarrow$ on multiplie à gauche et à droite par y_{ik}. Soit y_{ik}=0 auquel cas $x_{ij} = 0$. Soit $y_{ik} \neq 0$. Dans ce cas là il s'agit d'une constante, on le sort de la somme et on divise par y_{ik} à gauche et à droite.

$\sum_{i=1}^{n} x_{ij}$ - $\sum_{i=1}^{n} x_{ij} y_{ik} \leq M_i - M_i y_{ik}$, $\forall$ i,k, $\Leftarrow$ Découle des deux inégalités : $\sum_{j=1}^{n} x_{ij} y_{ik} \leq M_i y_{ik}$ et $\sum_{j=1}^{n} x_{ij} \leq M_i$.

$m_{ij} y_{ij} \leq x_{ij} \leq M_i y_{ij}$, $\forall$ i,j
$y \in \{0,1\}^{m \times n}$, $x \geq 0$

Et enfin, de sa relaxation (M) au modèle linéarisé (L).

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} (y_{ij} a_{ij} + c_{ij} x_{ij})$$

s.c

$\sum_{i=1}^{m} x_{ij} \geq A_j$, j $\in$ {1,...,n},
$\sum_{j=1}^{n} z_{jk}^{i} \leq M_{ij} y_{ij}$, $\forall$ i,j,k $\Leftarrow$ Réécriture du problème. On remplace $x_{ij} y_{ik}$ par z_{jk}^{i}.

$\sum_{i=1}^{n} x_{ij}$ - $\sum_{i=1}^{n} x_{ij} y_{ik} \leq M_i - M_i y_{ik}$, $\forall$ i,k,

$m_{ij} y_{ij} \leq x_{ij} \leq M_i y_{ij}$, $\forall$ i,j,
$z_{jk}^{i} \leq x_{ij}$ $\forall$ i,j,k, $\Leftarrow$ z_{jk}^{i}=$x_{ij} y_{ik}$ or $y \in \{0,1\}^{m \times n}$ donc $z_{jk}^{i} \leq x_{ij}$.

$y \in \{0,1\}^{m \times n}$, $x, z \geq 0$.

Ainsi, nous avons montré que les modèles sont équivalents.

2. **Vérifier que la valeur optimale de la relaxation continue du modèle (L) est au moins aussi bonne que la valeur optimale de la relaxation continue du problème originel (P).**

Les relaxations continues consistent à interpréter de façon continue un problème initialement discret. La variable discrete de notre problème était la variable y. Or $y \in \{0,1\}^{m \times n}$, donc $\forall i \in 1, 2, ..., m$ et $j \in 1, 2, ..., n$ alors $y_{ij} = 0$ ou $y_{ij} = 1$. On remplace cette contrainte stricte en une contrainte continue, c'est à dire $y \in \{0,1\}^{m \times n}$. Nous sommes donc maintenant dans le cas d'un problème d'optimisation continue.

On réécrit maintenant la relaxation continue de nos problèmes (L) et (P).

Relaxation continue du problème (P)

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{m}\sum_{j=1}^{n}(y_{ij}a_{ij} + c_{ij}x_{ij}) \\
\text{s.c} \quad & \sum_{i=1}^{m} x_{ij} \geq A_j,\ j \in \{1,...,n\} \\
& \sum_{j=1}^{n} x_{ij} \leq M_i,\ i \in \{1,...,n\} \\
& m_{ij}y_{ij} \leq x_{ij} \leq M_i y_{ij}, \forall i,j \\
& y \in [0,1]^{m\times n}, x \geq 0
\end{aligned}
$$

Relaxation continue du problème (L)

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{m}\sum_{j=1}^{n}(y_{ij}a_{ij} + c_{ij}x_{ij}) \\
\text{s.c} \quad & \sum_{i=1}^{m} x_{ij} \geq A_j,\ j \in \{1,...,n\} \\
& \sum_{j=1}^{n} z_{jk}^{i} \leq M_i y_{ij},\ \forall i,j,k \\
& \sum_{i=1}^{n} x_{ij} - \sum_{i=1}^{n} x_{ij}y_{ik} \leq M_i - M_i y_{ik}, \forall i,k \\
& m_{ij}y_{ij} \leq x_{ij} \leq M_i y_{ij}, \forall i,j \\
& z_{jk}^{i} \leq x_{ij} \forall i,j,k, \\
& y \in [0,1]^{m\times n}, x, z \geq 0
\end{aligned}
$$

Analyse des deux problèmes :

Nous cherchons à minimiser la même fonction : $\sum_{i=1}^{m}\sum_{j=1}^{n}(y_{ij}a_{ij} + c_{ij}x_{ij})$.
La contrainte $\sum_{i=1}^{m} x_{ij} \geq A_j,\ j \in \{1,...,n\}$ est commune aux deux équations.

On a $z_{jk}^{i} = x_{ij}y_{ik}$ par conséquent $\sum_{j=1}^{n} x_{ij} \leq M_i,\ i \in \{1,...,n\} \leftrightarrow \sum_{j=1}^{n} x_{ij}y_{ij} \leq y_{ij}M_i,\ i \in \{1,...,n\}$ et donc $\sum_{j=1}^{n} x_{ij} \leq M_i,\ i \in \{1,...,n\} \leftrightarrow \sum_{j=1}^{n} z_{jk}^{j} \leq M_i y_{ij},\ \forall i,j,k$

On conserve $z_{jk}^{i} = x_{ij}y_{ik}$ grâce à deux contraintes : $z_{jk}^{i} \leq x_{ij}\forall i,j,k$, et $z \geq 0$.
Les contrainte $m_{ij}y_{ij} \leq x_{ij} \leq M_i y_{ij}, \forall i,j$ et $y \in [0,1]^{m\times n}, x \geq 0$ sont présentes dans les deux équations.

Finalement toutes les contraintes de la relaxation continue de (P) sont conservées dans la relaxation continue de (L).Par contre, la contrainte $\sum_{i=1}^{n} x_{ij} - \sum_{i=1}^{n} x_{ij}y_{ik} \leq M_i - M_i y_{ik}, \forall i,k$ définie dans la relaxation continue de (L) n'est pas présente dans la relaxation continue de (P). Par conséquent, soit y et x de la valeur minimale de la relaxation continue du problème (P) ne satisfont pas $\sum_{i=1}^{n} x_{ij} - \sum_{i=1}^{n} x_{ij}y_{ik} \leq M_i - M_i y_{ik}, \forall i,k$, dans ce cas les valeurs minimales des deux problèmes seront identiques, soit x et y rentrent dans cette contraintes, dans ce cas, la relaxation continue du problème (L) ne peut pas prendre ces x et y.

Ainsi, la valeur minimale de ce problème pourrait être moins bonne que celle de la relaxation continue du problème (P) (donc supérieure).

On a donc dans tous les cas $opt^P \leq opt^L$.

Les expériences suivantes ont pour objectif de comparer les performances (temps de calcul et valeur de la solution trouvée) de quelques solveurs pour résoudre les modèles (P), (L) et (M).

Plus de détails vous seront donnés lors de la séance de suivi au sujet des solveurs choisis.

Dans le but de réaliser une comparaison de performance entre plusieurs solveurs et de réaliser plusieurs comparaisons sur différentes données, nous avons commencé par séparer les données des problèmes et les modèles.

Pour ce faire, nous avons créé plusieurs fichiers :

— un fichier **.mod** qui représente le modèle contenant ainsi la formulation générique du problème
— un fichier **.dat** qui lui est le fichier de données contenant les valeurs connues des paramètres du modèle

De cette manière, nous avons juste à changer le fichier **.dat** afin de tester le modèle avec d'autres paramètres.

Enfin, pour automatiser les commandes utilisées dans **ampl**, nous avons créé des fichiers **.run** qui contiennent toutes les commandes utiles à notre travail.

Comme les données des modèles fournis sont dans un fichier **.txt**, nous sommes passés par du code réalisé en **C++** afin de transformer automatiquement les fichiers **.txt** en format **.dat** avec la bonne syntaxe, et permettre ainsi un gain de temps de notre démarche de travail. Le fonctionnement de l'exécutable est simple, il suffit de passer un fichier **.txt** en entrée au format de notre problème, ainsi que le nom de fichier de sortie (notre .dat).

Nous avons aussi réalisé une deuxième version de ce convertisseur, le but est plus ou moins le même. Sa particularité est qu'il peut récupérer juste un partie des données. Ainsi, on obtient des fichiers de test avec des dimensions plus petites résolvant notre problème de dimension avec les solveurs. En effet, même avec une version complète **d'ampl**, pour un nombre de variables dépassant 180, le temps de calcul tend vers l'infini (ce problème est aussi peut être dû à la puissance de nos machines).

3. **Comparer les performances des solveurs intlinprog, cplex, gurobi et xpress sur les deux modèles (P) et (L). Pour ces expériences vous utiliserez AMPL (version commerciale de gmpl).**

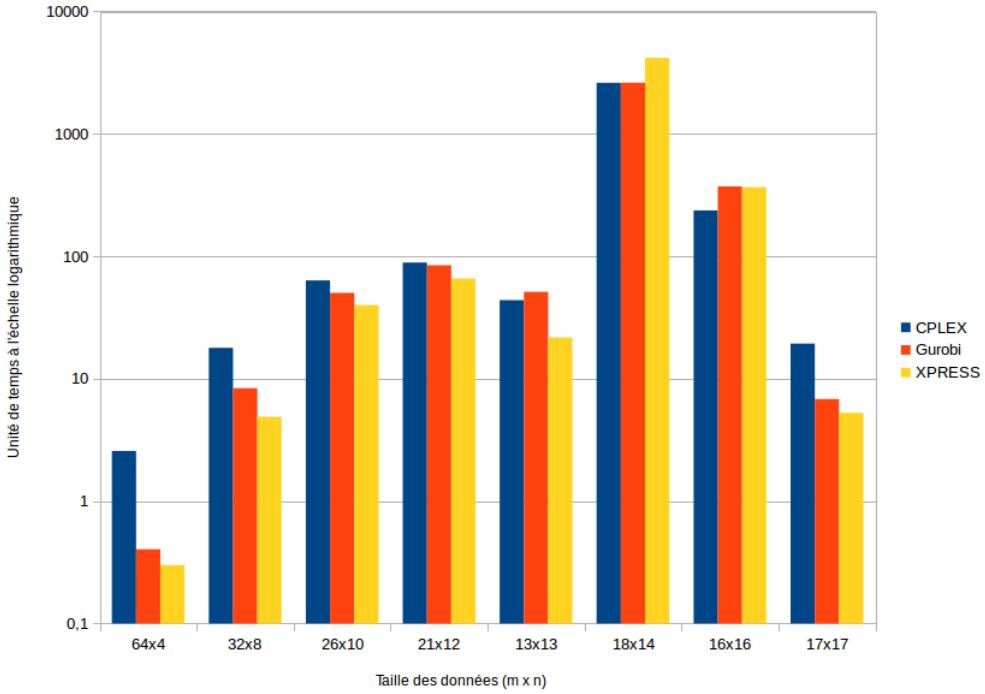
Considérons les problèmes (P) et (L).

Nous utiliserons les solveurs *cplex, gurobi et xpress* sur le logiciel AMPL pour les résoudre.

— **cplex** est un solveur créé par IBM, il est parmi les plus connus et les plus utilisés dans le monde pour des données de grande taille. Il résout les problèmes d'optimisation linéaires et quadratiques en continu ou à variables entières.
— **gurobi** est un solveur dont les performances se sont améliorées au cours des années, utilisé par de nombreuses entreprises internationales telles que Google ou Microsoft il reste une référence dans l'optimisation.
— **xpress** développé par FICO, est un solveur ayant fait ses preuves sur de nombreux problèmes à grande

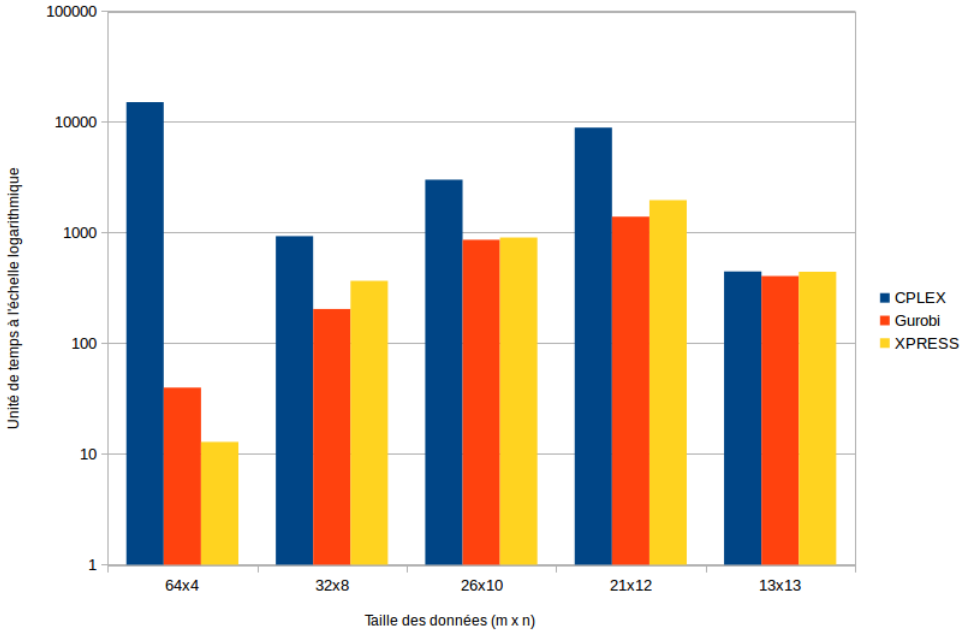
échelle. Il permet la résolution de problèmes instables ou difficiles grâce à son utilisation de la compression de données et de la matrice creuse.

Ces 3 solveurs ont put être testés sur un jeu de valeurs de tailles différentes. L'affichage du temps d'exécution permet de dresser cet histogramme comparatif pour le modèle P du problème tel qu'énoncé.



On lit dans ce graphique que pour différentes tailles de matrices, le solveur XPRESS semble le plus adapté pour ce problème. En effet, le temps d'exécution y est globalement moins long que pour d'autres solveurs.

On applique ensuite la même méthode mais pour le problème linéarisé.



On voit que le solveur Gurobi est cette fois-ci globalement plus performant que les autres, suivi de XPRESS.

On constate par ailleurs que la différence de temps entre le modèle P et L est toujours positive impliquant une performance supérieurement plus élevé avec le premier modèle quelque soit le solveur et les données utilisées.

4. **Comparer les performances des deux solveurs baron et knitro pour résoudre le modèle (M). Il s'agit d'un problème non linéaire en variables entières. Donc, un problème assez difficile.**

Considérons le modèle (M) suivant :

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} (y_{ij} a_{ij} + c_{ij} x_{ij})$$

s.c

$$\sum_{i=1}^{m} x_{ij} \geq A_j, \text{ j} \in \{1,...,\text{n}\},$$
$$\sum_{j=1}^{n} x_{ij} y_{ik} \leq M_i y_{ik}, \forall \text{ i,k}$$
$$\sum_{i=1}^{n} x_{ij} \text{ - } \sum_{i=1}^{n} x_{ij} y_{ik} \leq M_i - M_i y_{ik}, \forall \text{ i,k},$$

$$m_{ij} y_{ij} \leq x_{ij} \leq M_i y_{ij}, \forall \text{ i,j}$$
$$y \in \{0,1\}^{m \times n}, \; x \geq 0$$

Le but de cette partie est de comparer les performances de résolution des solveurs knitro et baron du modèle (M) qui est un problème non linéaire à variables entières.

Les solveurs Baron et Knitro :

— **Baron :**

Baron est un solveur non-linéaire général capable de résoudre les problèmes d'optimisation non-convexe à l'optimalité globale. Les variables de décision peuvent être continues, entières ou un mélange des deux. Ce genre de solveur est généralement utilisé pour des applications dans les industries des procédés chimiques, les produits pharmaceutiques, la production d'énergie, la conception technique et la gestion d'actifs.

— **Knitro :**

knitro est un solveur non linéaire particulièrement polyvalent, utilisant des options algorithmiques de pointe pour accommoder des fonctions non-linéarités et de contraintes variées dans des variables continues et entières. Il est conçu pour les problèmes à grande échelle avec des centaines de milliers de variables.

Ainsi, nous avons comparé les deux solveurs en utilisant les fichiers décrits plus haut. Cette comparaison s'est faite évidement sur les résultats de sortie (les valeurs de la solution) mais aussi sur le temps d'exécution. Ainsi, nous avons pu constater avec un nombre de tests réduits que le solveur **Baron** est bien plus rapide. Le tableau suivant, montre le temps (en secondes) de résolution du même problème par **Baron** et **Knitro** et ce en fonction du nombre de variables :

Nombre de variables	32	50	72	98	128	200
Temps de résolution Baron (secondes)	0.18	0.732	0.396	1.1	2.872	8.486
Temps de résolution Knitro (secondes)	0.312	13.308	83.28	212.72	344.8	1210.08
Solution fonction objectif (Baron)	559	1218	1094	1058	1134	NAN
Solution fonction objectif (Knitro)	559	1218	1094	1058	1134	NAN

FIGURE 2 – *Tableau de comparaison des performances des solveurs Baron et Knitro*

Troisième partie

Partie B : Résolution approchée

Pour cette partie, l'objectif est d'identifier un bonne méta-heuristique pour résoudre les problèmes (P) et (L). Une méta-heuristique sera qualifiée de bonne si elle retourne une solution réalisable de bonne valeur (pour le problème (P)) en un temps réduit.

5. **Comparer les performances des deux méta-heuristiques ga et patternsearch pour résoudre le problème (P).**

On cherche à comparer les performances de deux méta-heuristiques. Pour cela, on procède par étape. Toutes les données d'utilisations nous ont été donné par la documentation de Matlab

- **Étape 1 :** Représentation du problème. On crée 2 fonctions : une fonction objective donnant la fonction objective et une fonction contraintes permettant de stocker les contraintes sous la forme d'une matrice A et d'un vecteur b.

```
% La fonction objective
object= @(x) objective(x,m,n,a,c);  % a et c donnes par les scripts

% les contraintes
[A,b]=contraintes(AA,MM,mm_,m,n);
```

En effet, les contraintes seront modélisé par $Ax < b$. Notre vecteur x représentait donc nos valeurs $x_{i,j}$ écrites lignes par ligne dans un vecteur vertical et ensuite nos valeursy_{ij} écrites de la même façon. Le vecteur attendus

par notre fonction objectif est donc un vecteur colonne de $2mn$ valeurs.

La matrice A sera donc une matrice de $2mn$ colonnes. Son nombre de lignes sera le nombre de contraintes du problème étudié. Chaque ligne représente une contrainte telle que $A_{ij}x < b_i \ \forall j$ corresponde à la ième contrainte.

La création de cette matrice A se fait automatiquement grâce à notre fonction contraintes.

- **Étape 2 :** Implémentation de la méta-heuristique **ga**

On entre ensuite les variables nécessaires pour le fonctionnement de la fonction **ga**. On peut constater que nous avons *aplati* les variables, c'est-à-dire, que l'on a décidé de mettre les variables sous forme d'un vecteur plutôt que d'une matrice.

```
%Les variables
ub = ones(2*m*n,1)';  % y <= 1
ub(1:m*n) = Inf;  % x <= infini (documentation matlab)
lb = zeros(2*m*n,1)';  % x, y >= 0
```

Il suffit ensuite d'appeler la fonction **ga** en respectant les conditions énoncées dans la documentation et en ajoutant au préalable des options permettant le conditionnement de l'algorithme.

```
opti = gaoptimset('InitialPopulation',x0, 'TimeLimit',60, 'PopulationSize',
    3200,'PopulationType','doubleVector');

[X,fval,exitFlag,Output]=ga(object, 2*m*n, A,b,[],[],lb,ub,[],m*n+1:2*m*n,opti);
```

- **Étape 3 :** Implémentation de la méta-heuristique **patternsearch**

De même pour patternsearch :

```
X = patternsearch(object,x0,A,b,[],[],lb,ub);
```

Remarque : patternsearch semble ne pouvoir être utilisé que pour les variables continues et non entières contrairement à ga qui fonctionne pour les deux. Nous avons ainsi décidé de prendre les variables continues obtenues par patternsearch et de créer un vecteur en variables entières respectant les contraintes.

- **Interprétation :** Dans un premier temps, afin de vérifier nos résultats, nous avons décidé de lancer le même jeu de données sur AMPL et en Matlab. Pour cela, on choisit une valeur de x0 réalisable reposant sur le vecteur AA et la matrice mm_ données. Puis, nous lançons une partie du jeu de données du fichier **1.txt**

se trouvant dans le dossier **1** par soucis de temps de fonctionnement du programme. Nous obtenons alors une solution de la fonction objectif de **559** sous Baron et Knitro, d'une solution de la fonction objectif d'environ **900** sous ga et d'environ **250** sous patternsearch en prenant les valeurs continues et **800** avec la transformation en valeurs entières.

Ainsi, le résultat en valeurs continues de 250 sous patternsearch s'explique par le fait qu'il fonctionne sous valeurs continues. Les valeurs étant continues, il y a plus de solutions possibles et donc la probabilité de trouver une valeur de la fonction objective plus petite est plus importante. Cependant, la valeur de ga nous semble étrange. De plus les valeurs du vecteur X renvoyé par ga sont extrêmement proches des valeurs initialisées dans le vecteur x0. Les valeurs sembleraient "bloquées" dans des minimaux locaux. Nous décidons donc dans un second temps de lancer les calculs sur toutes les données du fichier 1.txt. Nous obtenons cependant toujours un résultat meilleur avec Patternsearch et des valeurs de ga bloquées dans les minimums locaux.

Ainsi, en se basant sur le résultat de la fonction objectif sur le temps d'exécution des deux métaheuristiques, on trouve que **Patternsearch est la plus performante**. Il s'agira donc de la métaheuristique que nous utiliserons dans la partie C.

6. **Comparer les performances des deux méta-heuristiques ga et patternsearch pour résoudre (P) en les initialisant avec la solution de la relaxation continue du problème (L) sans les contraintes (10).**

L'objectif de cette question est de voir si les résultats trouvés précédemment vont être modifiés en initialisant avec la solution de la relaxation continue du problème (L) sans les contraintes (10). Nous implémentons ainsi les contraintes manquantes. Pour pouvoir tester la question 6 ou 7, il suffit de décommenter les contraintes correspondantes. Ici un phénomène étrange à lieu, le programme ne semble pas vouloir s'arrêter. Même en imposant un temps limite de 60 secondes, il continue à tourner.

7. **Comparer les performances des deux méta-heuristiques ga et patternsearch pour résoudre (P) en les initialisant avec la solution de la relaxation continue du problème (L) sans les contraintes (11).**

La question 7 étant similaire à la question 6, les conclusions sont les mêmes.

Quatrième partie

Partie C : Algorithme de sous gradient

Cette dernière partie à pour objectif de résoudre le problème (L) à l'aide de la technique de relaxation lagrangienne.

Il existe plusieurs relaxations lagrangiennes du problème (L). Ces relaxations se distinguent par les contraintes dualisées. Nous vous proposons d'en tester deux de ces relaxations. Vous pouvez, bien sûr, en comparer d'autres.

Vous utiliserez dans l'algorithme de sous-gradient la meilleure des deux heuristiques identifiée dans la partie précédente, le pseudo-algorithme de sous gradient étant le suivant :

Algorithm 1 A subgradient method to optimize the Lagrangian dual problem

Require:
- ε control parameter
- $\rho \in]0, 2[$ control parameter
- π^0 initial nonnegative vector (usually the nul vector)
- `iterLimit` number of maximum iterations
- `DualNoChangTOL` accepted number of iterations without change in the value of the dual function

Ensure: A local optimum of θ and an approximate primal solution $\hat{x}$

Set $k \leftarrow 1$ and $t \leftarrow 1$
Set $\beta^k \leftarrow -\infty$ and $\pi^k \leftarrow \pi^0$
repeat
  Solve the Lagrangian relaxation : $\theta\left(\pi^k\right) \leftarrow \min_{x \in \mathcal{F}} L\left(x, \pi^k\right)$
  Set $x^k \leftarrow \arg\min\left\{L\left(x, \pi^k\right) : x \in \mathcal{F}\right\}$
  for all $j \in J$ **do**
    $\gamma_j^k \leftarrow g_j\left(x^k\right)$
  end for
  if $\theta\left(\pi^k\right) > \beta^k$ **then**
    Set $\beta^{k+1} \leftarrow \theta\left(\pi^k\right)$
  else
    if $t \leq$ `DualNoChangTOL` **then**
      Set $t \leftarrow t + 1$
    else
      Set $\rho \leftarrow \frac{\rho}{2}$
      Set $t \leftarrow 1$
    end if
  end if
  if $\gamma^k = 0$ **then**
    `Stop; The current solution is optimal`
  else
    $\hat{\theta} \leftarrow$ `applyMHeuristic`$\left(x^k, f, \mathcal{X} \cap \mathcal{F}\right)$
    for all $j \in J$ **do**
      $\pi_j^{k+1} \leftarrow \max\left\{0, \pi_j^k - \left(\rho \frac{\theta\left(\pi^k\right) - \hat{\theta}}{\|\gamma^k\|}\right)\gamma_j^k\right\}$
    end for
    Set $k \leftarrow k + 1$
  end if
until $\frac{|f(x^k) - \theta(\pi^k)|}{\theta(\pi^k)} \leq \varepsilon$ or $k >$ `iterLimit`

Afin de répondre aux questions suivantes, nous avons tout d'abord dû implémenter cet algorithme de sous-gradient. Nous allons donc détailler ici le fonctionnement de notre code.
Comme expliqué dans le pseudo-code, nous avons tout d'abord initialisé nos variables :

```
1 k=1;
```

```
t=1;
beta=-inf;
pik=pi0;
```

Puis, après être entré dans la boucle, on résout la relaxation Lagrangienne :

```
f0= vecteurLagr56(m,n,a,c,pik,MM);
[A,b]=contraintesLF( AA, MM, mm_, m, n);
[x, theta] = linprog(f0, A, b, [], [], lb, ub);
```

La fonction **vecteurLagr56** renvoie un vecteur correspondant à la fonction de Lagrange associée au problème L et ne prenant en compte que la contrainte (56). D'autres parts, la fonction **contraintesLF** correspond aux contraintes F souhaitées. Pour finir, on résout la relaxation Lagrangienne de notre problème grâce à **linprog** disponible sous matlab.
Cela correspond à trouver $\theta(\pi)$ et π tels que :

$$\theta(\pi) = min\{L(x,\pi) : x \in F, \pi \geq 0\} \tag{1}$$

Où

On va maintenant s'intéresser à la partie du code traitant la solution trouvée précédemment :

```
if gamma==zeros(J,1)
      fprintf("gamma=0\n");
      break;
else
     [A,b]=contraintesL(AA, MM, mm_, m, n);
     f2=objectivevec(m,n,a,c);
     [x1, theta_c] = linprog(f2,A,b,[],[],lb,ub);
     for i=m*n+1:2*m*n
        if(x1(i)~=0)
           x1(i)=1;
        end
     end
     theta_c=f2'*x1
     for j=1:J
       pik(j) = max(0,pik(j)-(rho*(theta-theta_c)/norm(gamma))*gamma(j));
     end
     k= k+1;
end
```

La première chose à vérifier est la valeur de γ, en effet, si ce vecteur est nul, la solution est optimale et on sort de la boucle. Si cela n'est pas le cas, on remet à jour les contraintes ainsi que la fonction objective puis on résout le nouveau problème avec **linprog**.

La fin du code représente seulement la condition d'arrêt :

```
if(abs( object(x) - theta )/theta <= eps || k > iterLimit)
  if(k > iterLimit)
    fprintf("limite atteinte\n");
  else
    fprintf("Valeur optimale\n");
  end
  break;
end
```

8. **Résoudre à l'aide d'un algorithme de sous-gradient la relaxation continue du problème (L) en dualisant seulement les contraintes (10). Vous utiliserez la meilleure méta-heuristique pour identifier une solution réalisable entière du problème (P).**

9. **Résoudre à l'aide d'un algorithme de sous-gradient la relaxation continue du problème (L) en dualisant seulement les contraintes (11). Vous utiliserez la meilleure méta-heuristique pour identifier une solution réalisable entière du problème (P).**

10. **Commenter vos résultats**

Bibliographie

[1] Polytech Sorbonne, "Logo Polytech Sorbonne." (https://www.facebook.com/pg/PolytechParisUPMC/photos/?tab=album&album_id=156421134394814).

[2] Sorbonne Universite, "(Logo Sorbonne Universite." (http://www.sorbonne-universites.fr).

[3] Image Couverture, "Image couverture." (https://upload.wikimedia.org/wikipedia/commons/thumb/1/12/Optimisation\Lineaire.svg/330px-OptimisationLineaire.svg.png).

[4] AMPL, "Ampl Solver." (https://ampl.com/products/).

[5] EPFL, "Cours Optimisation." (https://transp-or.epfl.ch/optimization/slides/10-integer.pdf).