Université Gustave Eiffel

Le 29/11/2022
M1 Informatique

Semestre 7

Compte Rendu : LP SOLVE

Binôme:

MENAA Mathis
REGUEME Yohann

Encadrants:

M.THAPPER

Répartition du travail : *Lors de ce mini-projet, nous avons effectué tout le travail produit ensemble.*

Introduction

Dans le contexte du cours d'optimisation, nous devons résoudre et mettre en pratique un problème d'optimisation ainsi que l'utilisation de LP-Solver.

Ce rapport répertorie les réponses aux questions posées dans le sujet, sur les conclusions que nous tirons de nos résultats ainsi qu'une explication des algorithmes utilisés.

Problème générique

1. Exercice 4

Pour traiter le problème, on crée une classe ProblemToLp qui répertorie l'ensemble des méthodes que l'on va pouvoir appliquer à notre fichier d'entrée pour le convertir en fichier LP_Solve.
Tout d'abord, on récupère depuis notre fichier d'entrée les variables **m** (types de ressources) , **n** (types de produits), les **contraintes** des produits et leurs **valeurs** pour chacun des produits.
Avec ces données on a écrit une méthode pour chaque étape de construction du fichier de sortie (Fonction Objectif, Équations de contraintes, Gestion de l'option -int..).

2. Exercice 5

On a créé une méthode dans notre classe qui va exécuter le programme **.lp** créé précédemment et l'afficher.
Pour exécuter la commande on a utilisé **subprocess.run**, qui nous récupère aussi l'output de la commande.
On met en forme l'output que l'on affiche à la fin de la méthode.

3. Exercice 6

Test de notre programme sur le fichier data.txt :

a) *Bénéfice optimal : 21654.79332331*
b) *Nombre de produits différents fabriqués : 6 produits différents*
c) *Temps d'exécution mesurée grâce à time :*

real 0m0,014s
user 0m0,000s
sys 0m0,008s

4. Exercice 7

Test de notre programme sur le fichier data.txt avec l'option -int :

a) *Bénéfice optimal : 21638.00000000*
b) *Nombre de produits différents fabriqués : 12 produits différents*
c) *Temps d'exécution mesurée grâce à time :*

real 0m20,073s
user 0m20,062s
sys 0m0,004s

5. Exercice 8

Test de notre programme sur le fichier bigdata.tx :

a) *Bénéfice optimal : 5050533.10609334*

b) *Nombre de produits différents fabriqués : 100*

c) *Temps d'exécution mesurée grâce à time :*

real 0m1,097s
user 0m0,992s
sys 0m0,038s

Test de notre programme sur le fichier bigdata.txt avec l'option -int :

Après plusieurs minutes d'attente on remarque que le programme ne se termine pas. Comme vu en cours, la programmation linéaire en nombres entiers est un problème NP-difficile.

6. Mode d'emploi : generic.py

Entrée de notre programme : generic.py est un programme prenant en entrée un fichier formaté décrivant un programme linéaire, le nom du fichier de sortie .lp contenant le fichier d'entrée sous format.lp ainsi qu'une option (-int) ajoutant les contraintes d'intégralité sur toutes les variables .

Sortie de notre programme :

- Sur la sortie standard, le résultat de l'exécution du fichier **.lp** précédemment généré.
- Un output **.lp** décrivant le problème formaté dans le fichier d'entrée.

Commande de lancement :

```
python3 generic.py <input.txt> <output.lp> [OPTION]
```

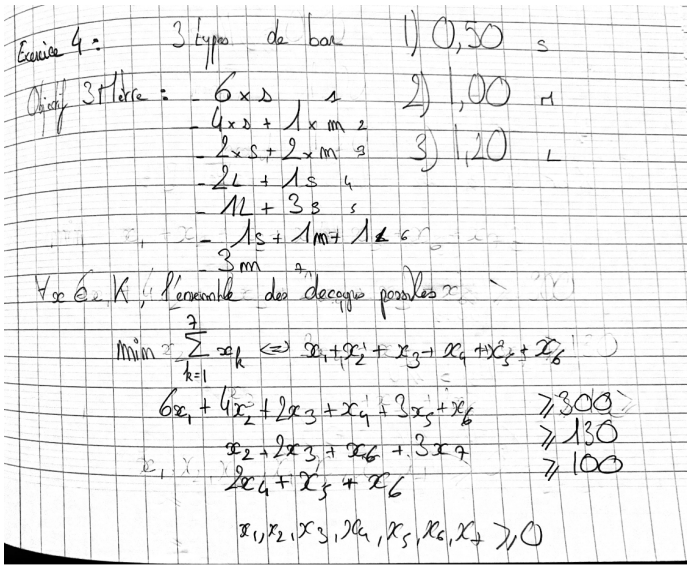
Le résultat est de la forme suivante (sans l'option) :

```
$ python3 generic.py exemple1.txt output.lp
opt = 3250
P1 = 15
P2 = 10
```

- Première ligne, optimum trouvé avec sa valeur.
- Les lignes suivantes présentent les différents produits ainsi que leurs nombres.

Problème de découpe

1. Exercice 9



Comme vu en TD, on retrouve le programme linéaire ci-dessus avec 7 découpes différentes avec la liste [50,100,150].

2. Exercice 10

Pour l'algorithme de notre fonction, on va utiliser la récursivité pour pouvoir obtenir toutes les découpes possibles en donnant une liste de longueurs souhaitées ainsi que la longueur d'une barre de base. On commence tout d'abord par calculer le nombre de barres de taille max qu'on peut utiliser, notre condition d'arrêt est que si il ne reste plus qu'une taille de barre dans la liste alors on retourne ça *size_possible* qui détermine combien de barre de cette taille on peut placer. La récursivité nous permet ensuite d'obtenir toutes les combinaisons possibles de longueurs de barres pouvant combler la taille objectif.

Avec les entrées : 500 et [200, 120, 100, 50]

On obtient la liste de découpe suivante : [[0, 0, 0, 10], [0, 0, 1, 8], [0, 0, 2, 6], [0, 0, 3, 4], [0, 0, 4, 2], [0, 0, 5, 0], [0, 1, 0, 7], [0, 1, 1, 5], [0, 1, 2, 3], [0, 1, 3, 1], [0, 2, 0, 5], [0, 2, 1, 3], [0, 2, 2, 1], [0, 3, 0, 2], [0, 3, 1, 0], [0, 4, 0, 0], [1, 0, 0, 6], [1, 0, 1, 4], [1, 0, 2, 2], [1, 0, 3, 0], [1, 1, 0, 3], [1, 1, 1, 1], [1, 2, 0, 1], [2, 0, 0, 2], [2, 0, 1, 0]]

Ce qui nous donne au total 25 découpes.

3. Exercice 11

Comme pour le précédent problème, notre programme est constitué de la classe ***cutToLP*** ainsi que de la fonction ***get_all_optimized_cut*** écrite précédemment.
Notre programme va se charger de calculer toutes les coupes possibles, pour ensuite pouvoir modéliser le problème dans un fichier .lp et ensuite exécuter LPSolve sur ce fichier. Il affichera dans le terminal l'optimum trouvé ainsi que la valeur optimale pour les différents découpages.

7. Mode d'emploi : cut.py

Entrée de notre programme : cut.py prend en paramètre, la taille d'une barre, la liste des différentes taille voulues ainsi que la liste du nombre de barre voulue pour chaque taille.

Sortie de notre programme :

- Sur la sortie standard, le résultat de l'exécution du fichier **.lp** précédemment généré.
- Un **myCut.lp** décrit le problème formaté dans le fichier d'entrée.

Commande de lancement :

```
python3 cut.py <SIZE> <SIZE_LIST> <NB_OF_EACH_WANTED>
```

(exemple sous windows : python3 .\cut.py 500 "[200 ,120, 100,50]" "[60,100, 150, 350]")

- SIZE : la taille d'une barre
- SIZE_LIST : liste des découpes de barres voulus
- NB_OF_EACH_WANTED : liste du nombre de découpe de SIZE_LIST voulus.

Conclusion

Ces différents exercices nous auront permis de modéliser en informatique, grâce à **python** ainsi qu'a **lp_solve** les programmes linéaires vu en cours et en TD.
Nous avons appris à adapter **lp_solve** à **python** en concevant un programme permettant de formater un fichier représentant un programme linéaire et d'ensuite pouvoir le résoudre grâce à **lp_solve**.