Chen Steve - Van Heusden Nicolas (Groupe apprentis)

Chargé de TD : Thapper Johan

TP1 : Optimisation lp_solve

1 Prise en main de lpsolve

Exercice 1

Tout fonctionne bien. Les valeurs optimales de x1 et x2 sont 2 et 8 respectivement.

Exercice 2

Oui.

Exercice 3

On retrouve bien le même résultat que dans le TD1 exo1.

2 Un problème générique

Exercice 4 & 5

Guide utilsateur : pour lancer le programme taper la commande suivante :

```
python3 generic.py le_nom_du_fichier_txt le_nom_du_fichier_lp
```

par exemple avec la commande suivante :

```
python3 generic.py data/exemple1.txt exemple1.lp
```

le programme affiche :

```
time :  0.005711078643798828 secondes
P1 = 20
P2 = 15
opt = 4500.00000000
```

Le time correspond donc au temps d'éxécution du programme lp_solve.

P1, P2 correspondent aux nombre d'éléments max que l'on doit prendre dans chaque variable.

opt correspond à l'optimum.

Exercice 6

(a) Le bénéfice optimal est **21654.79332331**

(b) Il faut fabriquer **5 produits différents**

(c) Le solveur a pris **0.0066394805908203125 secondes** pour calculer la solution optimal

Exercice 7

Avec les contraintes d'intégralité sur le fichier data.txt nous obtenons : **(a)** Le bénéfice optimal est **21638.00000000**

(b) Il faut fabriquer **7 produits différents**

(c) Le solveur a mis **24.85676383972168 secondes**

Les contraintes d'intégralité qui force les variables à prendre des valeurs entières rend le problème plus complexe. Il faut pour cela séparer le problème afin d'éliminer les solutions fractionnaires, ces séparations donnent naissance à un arbre chaque noeud correspond à la résolution d'une portion du problème original. C'est pourquoi temps d'exécution croit exponentiellement. Cependant il apporte un meilleur résultat qu'une optimisation linéaire classique.

Exercice 8

Sans les contraintes d'intégralité sur le fichier bigdata.txt nous obtenons :

(a) Le bénéfice optimal est **5050533.10609334**

(b) Il faut fabriquer **3000 produits différents**

(c) Le solveur a mis **1.810274362564087 secondes**

Avec les contraintes d'intégralité sur le fichier bigdata.txt nous ne parvenons pas à obtenir un résultat. A cause des contraintes d'intégralité (expliqué dans la question précèdente).

Exercice 9

Pour minimiser le nombre utilisé, il faut : découper une barre en 6 morceaux de 50 cm 20 fois. découper une barre en 2 morceaux de 50 cm et 2 morceaux de 100 cm 65 fois découper une barre en 1 morceau de 50 cm et 2 morceaux de 120 cm 50 fois.

en tout on utilisera 135 barres.

Exercice 10 - 11

Guide utilisateur: pour lancer le programme, tapez la commande suivante :

```
python3 decoupe.py 500 60 200 100 120 150 100 350 50
```

Le premier argument est la longueur de barre donnée en cm. Les arguments suivant fonctionnent par paire : Le premier est le nombre d'éléments qu'on veut Le second est la taille de l'élément en question

S'il manque une valeur sur la ligne de commande alors le programme indique :

```
You must give 2 elements by 2
```

Sinon le programme affiche :

```
une solution optimale utilise 114 barres de longueur 500 cm.
```

Algorithme de génération des découpages

Pour générer les différents découpages maximaux, nous avons réaliser une fonction récursive prenant en paramètre la longueur des barres, une liste de longueurs souhaitées ainsi qu'une liste initialisé à vide par défaut (elle contiendra notamment tous les découpages possible).

Tout d'abord cette fonction genère une solution de découpage maximale qui sera ajouté à la liste de solutions de découpages maximaux si elle n'y est pas déjà. Puis on effectue un appel récursive en affectant la liste par défaut avec la nouvelle liste contenant les solutions.