Université Gustave Eiffel

Master Informatique

Rapport Technique

Optimisation

Jimmy Teillard

Groupe 2

Chargé de TD: Anthony Labarre

1ère Année

Décembre 2021

Contents

1 Introduction

Ce rapport technique à pour but de compiler mes résultats quant aux exercices du TP lpsolve.

Chacune des parties du TP sera traitée dans une section dédiée, avec les résultats et réponses aux questions posées, en plus d'un mode d'emploi du script créé.

2 Un problème générique

2.1 Mode d'emploi

Utilisation : `python3 generic.py [-int] input.txt output.lp`

Pour lancer le script, il est nécessaire de fournir un fichier en entrée (ex: `input.txt`) formaté correctement, ainsi qu'un nom de fichier de sortie où écrire la modélisation d'un programme linéaire (ex: `output.lp`)

Il est possible d'ajouter une option `-int` afin d'ajouter une contrainte d'intégrité sur chacune des variables.

Voici un exemple de fichier d'entrée valide :

```
3 2
350 750 600
P1 10 30 10 150
P2 10 10 20 100
```

La première ligne contient deux valeurs :

- `m`: le nombre de ressources
- `n`: le nombre de produits

La deuxième ligne contient `m` valeurs qui correspondent chacune à la limite de chaque ressource.

Les lignes suivantes représentent chacun les contraintes desdits produits. Pour chaque ligne, le premier bloc est le nom du produit, suivit de `m` coûts pour chaque ressource (dans l'ordre), terminée par le bénéfice de production de ce produit.

Lancer le script permet de lancer lpsolve sur ces données ; le résultat est formaté de la façon suivante :

- Chaque produit et leur quantité optimale (par ligne) (si la quantité est 0, le produit n'apparaît pas.
- La valeur d'optimum atteinte
- Le nombre de produits différents utilisés

2.2 Exercice 4

Pour un fichier donné en entrée, mon algorithme récupère d'abord la première ligne afin d'extraire les valeurs `m` et `n` car elles sont nécessaires pour savoir combien d'itérations sont nécessaires. Je construit ensuite une liste de `Products`, chacun avec un nom, une liste de coûts, et un gain.

Grâce à ces composants, je peux construire le contenu du fichier lp ligne par ligne itérativement.

2.3 Exercice 5

Afin de récupérer le résultat de lpsolve pour le formater correctement, j'ai utilisé la fonction `popen(cmd)` du module `os`, car il est alors possible de traiter son retour exactement de la meme façon qu'un fichier classique (avec `with ... as ...:` et `readLines`)

2.4 Exercice 6

Pour le fichier `data.txt`:

- Le bénéfice optimal atteint est d'environ 21654.79
- 6 produits différents sont fabriqués
- Le calcul (traitement du fichier compris) a pris environ 0.05s

2.5 Exercice 7

Pour le fichier `data.txt`, avec contrainte d'intégralité:

- Le bénéfice optimal atteint est de 21638
- 12 produits différents sont fabriqués
- Le calcul (traitement du fichier compris) a pris environ 1 minute et 40.85s (PC fatigué)

On se rend compte que l'ajout de la contrainte d'intégralité change non seulement drastiquement les résultats obtenus, mais aussi le temps de calcul. En effet, lpsolve calcule ces données en essayant à tatons les valeurs des variables pour se rapprocher de la solution, c'est essentiellement du bruteforce.

2.6 Exercice 8

Pour le fichier `bigdata.txt`:

- Le bénéfice optimal atteint est d'environ 5050533.11
- 100 produits différents sont fabriqués
- Le calcul (traitement du fichier compris) a pris environ 1.417s

Malgré la quantité colossale de données, le calcul de la solution était beaucoup rapide que celui de la question précédente.

2.7 Exercice 9

Pour le fichier `bigdata.txt`, avec contrainte d'intégralité, le résultat n'est jamais atteint...

De la même façon que pour la question 7, le calcul prend un temps plus long, voire même effectivement infini dans ce cas-ci: Le problème est NP-difficile.

3 Un problème de découpe

3.1 Exercice 9

Une modélisation possible du problème de l'exercice 4 du TD 2 (les barres d'aluminium) est la suivante:

```
min: x1 + x2 + x3 + x4 + x5 + x6 + x7;
6 x1 + 4 x2 + 3 x3 + 2 x4 + x5 +   x6        >= 300;
         x2        + 2 x4 + x5        + 3 x7 >= 130;
                x3        + x5 + 2 x6        >= 100;
int x1 x2 x3 x4 x5 x6 x7;
```

Où chaque variable x_n représente une coupe optimale possible d'une barre de 300cm en barres de 50cm, 100cm, et/ou 120cm. Chaque contrainte représente le nombre minimum de chaque type de barre requis (respectivement 300, 130, et 100 unités). Nous voulons minimiser le surplus de chaque découpe.

Le plan de découpe optimal est le suivant:

Soient `s` (small) une barre de 50cm, `m` (medium) une barre de 100cm, et `l` (large) une barre de 120cm.

- 41 découpes `6s`
- 1 découpe `4s, 1m`
- 50 découpes `1s, 2l`
- 43 découpes `3m`

3.2 Exercice 10

Mon algorithme de recherche des découpes optimales est le suivant:

Pour entrées `max`, un entier, qui est la taille d'une barre, et `sizes` les tailles possibles de découpe dans l'ordre décroissant.

Prendre la première taille de `sizes` (soit `size`) et récupérer le nombre maximal que l'on peut en couper dans `max`, soit `maxcut`. Si `sizes` est de longueur 1, alors renvoyer `[[maxcut]]` ; sinon, pour chaque quantité de 0 à `maxcut`, appeler récursivement l'algorithme pour `max` étant la différence du `max` actuel et du produit de la `size` récupérée et de la quantité actuelle, et `sizes` les tailles suivantes.

Effectivement, cet appel récursif se résumé à réduire la taille max de la barre à utiliser en consommant des barres de plus en plus petites.

En récupérer les valeurs, et les ajouter à la liste finale.

Pour des barres de 500cm, et des tailles possibles de 200cm, 120cm, 100cm, et 50cm, il y a 25 découpes optimales possibles.

3.3 Exercice 11 — Mode d'emploi

Le programme `steelcut.py` prends les arguments suivants:

- max: la taille d'une barre d'aluminium
- sizes: une liste séparé par des virgules des tailles de coupe possibles dans l'ordre décroissant
- amounts: les quantités minimums requises de production de chaque barre coupée
- output file: le fichier où écrire le programme linéaire

Le programme répond alors le nombre de barres d'aluminium requises pour répondre aux contraintes de quantités.

4 Conclusion

Ce TP m'a permis de prendre connaissance et de manipuler l'outil lpsolve, tout en élaborant un algorithme intéressant de recherche de coupes optimales.

Les résultats obtenus sont en concordances avec ce que l'on a pu voir en cours.