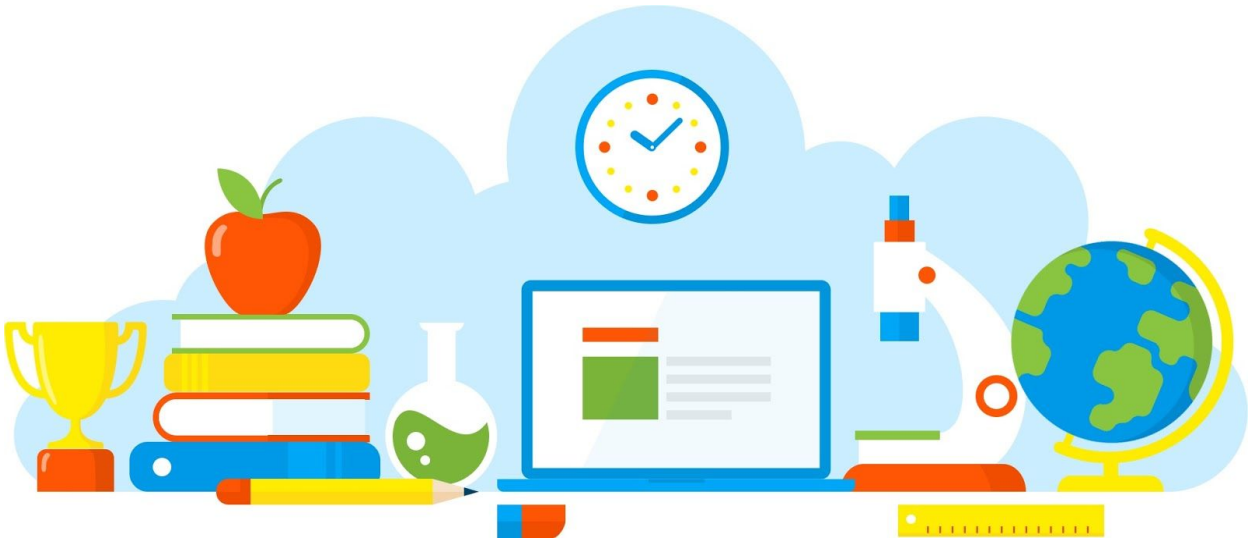
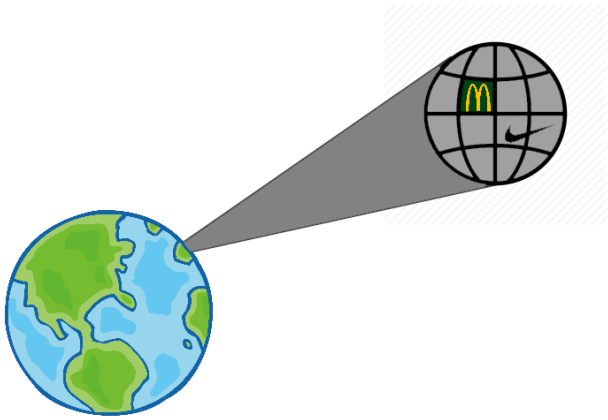


# Rapport Moon

*Yoan ROCK -- Axel Morvan*



10/01/2020

# Sommaire :



# Introduction :

## Sujet

Ce projet consiste en l'étude du problème MOON.

Une entreprise possède la surface de la Lune et elle souhaite vendre des parties de cette surfaces pour y poser des annonces publicitaires. Chaque client choisit la quantité de surface qu'il souhaite acheter et le prix qu'il prêt à payer. C'est le vendeur qui doit décider à qui et quelle surfaces il va vendu.

Tout ça est possible car la Lune montre toujours la même face à la Terre et un éclairage spécial a´été mis en place pour que la surface soit visible toutes les nuits de l'année  (Le cas de la météorite passant entre la Terre et la lune n'a pas encore été étudié).

Voici quelques contraintes :

- Une case ne peut pas être vendue à 2 annonceurs.

- On n'est pas contraint de vendre toutes les cases de la Lune (il peut donc y avoir des cases vendues, et donc il peut y avoir plus de cases à vendre que demandées par les annonceurs.
- On n'est pas contraint de vendre à tous les annonceurs. (Et donc il peut y avoir plus de cases à demandées par les annonceurs que de cases à vendre.

L'objectif est de maximiser les gains de ventes des cases de la Lune.

## Contribution

Axel c'est occupé de l'étude de la complexité, du programme linéaire et de l'évaluation des performance. Tandis que Yoan c'est plutôt occupé de l'algorithme exacte ( et de l'étude de sa complexité) , de la modélisation du problème, de l'introduction et de la conclusion du projet.

# Modélisation :

Soit une grille de taille **WxH**. un entier n

Il y a **n** triplet **(Mi,Hi,Wi)** $i \in [1,n]$ ou Mi,Hi et Wi sont des entiers tels que :

$$\begin{cases} 1 \leq Hi \leq H \\ 1 \leq Wi \leq W \\ 0 \leq Mi \end{cases}$$

Un ensemble $I \in [1,n]$ qui corresponds aux annonceurs sélectionnés

$\forall\ i \in I$

il y a 2 entier **Li** et **Ci** qui correspondent respectivement à la ligne et la colonne en haut à gauche du rectangle dans la grille. tels que :

$$\begin{cases} 1 \leq Li \leq w\text{-}Wi+1 \\ 1 \leq Ci \leq Hi+1 \end{cases}$$

tels que $\forall\ i,j \in I$ et ou $i \neq j$ :

$Cj \geq Ci + Wi$ et $Ci \geq Cj + Wj$

$Lj \geq Li + Hi$ $Li \geq Lj + Hj$

La fonction objectif est la suivante : $\sum_{i \in I} \text{Min}\left(M_i, \sum_{l=l_i}^{l_i+H_{i-1}} \sum_{c=c_i}^{C_i+w_{i-1}}\right)$

## Etude de la complexité :

Mq $(MOON)_D$ est NP-Complet

$(MOON)_D \in NP$ car $(MOON) \in NPO$ et il existe une combinaison de certificat/vérificateur.

1) Mq $(RECTPACK)_D \leq (MOON)_D$

   $(RECTPACK)_D$: Soit une grille $H * W$ et $n$ rectangles de taille $H_i * W_i$. Est-il possible de placer tous ces rectangles dans la grille sans recouvrement ?

   Soit le problème $(MOON)_D$ avec une instance $I$ telle que:

   - $n$ annonceurs
   - $H_i = W_i = 1, \forall i \in [[1; n]]$ leurs tailles associées
   - $\omega_i = +\inf$ leurs coûts max associés

   Alors $I$ est aussi une instance de $(RECTPACK)_D$
   Donc, $(RECTPACK)_D \leq (MOON)_D$

2) Mq $(RECTPACK)_D$ est NP-Difficile

   Soit $(BINPACK)_D$ NP-Difficile au sens fort
   Mq $(BINPACK)_D \leq (RECTPACK)_D$

   $(BINPACK)_D$: Soit $x_1, , x_n$ entiers et $K \in N$. Peut-on utiliser $K$ sacs de taille $B$ pour contenir tous les objets $x_i$ ?

   Soit le problème $(RECTPACK)_D$ avec une instance $I'$ telle que:

   - $K$ rectangles de taille $H_i = 1$ et $W_i = n, n \leq B$

Alors $I'$ est aussi une instance de $(BINPACK)_D$

Donc, $(BINPACK)_D \leq (RECTPACK)_D$

Donc $(BINPACK)_D \leq (RECTPACK)_D \leq (MOON)_D$

Or $(BINPACK)_D$ est NP-Complet, donc $(MOON)_D$ est NP-Difficile, donc NP-Complet car $(MOON)_D \in NP$.

## Algorithme exact :

Résumé : l'algorithme exact itère sur toutes combinaisons possibles d'agencement de la lune. puis pour chaque instance il calcul le gain obtenu puis les comparent tous pour finalement avoir le meilleure agencement possible et le meilleure bénéfice. Nous utilisons

**On instancie un tableau de grille vide T**

**| pour chaque case de la grille, on essaye de placer l'annonceur si c'est possible**

**| Si l'annonceur à été placé**

**| récursivement on continue à essayer de positionner les autres annonceurs**

**| On ajoutes la grille dans T**

**On retourne T**

**Pour chaque grille dans T**

**| On calcul le gain obtenu par l'agencement des annonceurs**

**| On les gains obtenus pour chaque grille et on choisi la grille avec le gain optimal**

**On affiche la grille choisie**

**On affiche le gain associé**

La fonction est récursive , la fonction s'appelle pour toute les cases pour trouver toutes les combinaison de l'agencement de tous les annonceurs sur toutes les cases ( sans oublier le cas où l'annonceur n'est pas positionné du tout ) donc sa complexité est :

**O(** $(wh+1)^{\underline{n}}$ **) = O(** $2^n$ **). La complexité est donc exponentielle.**

ou wh est la taille de la grille et n le nombre d'annonceurs

## Programmation linéaire :

Le programme linéaire permet de résoudre le problème plus rapidement et plus facilement. Tout d'abord, il a fallu déterminer la fonction objectif, les variables et les contraintes:

Fonction objectif:

- $max(\sum_{a \in A} \sum_{l=0}^{h-H(a)-1} \sum_{c=0}^{w-W(a)-1} w_{a,l,c} * y_{a,l,c})$

Variables:

- $y_{a,l,c} = \begin{cases} 1 & \text{si l'annonceur est placé} \\ 0 & \text{sinon.} \end{cases}$
- $w_{a,l,c} = min(M(a); \sum_{i=l}^{l+H(a)-1} \sum_{j=c}^{c+W(a)-1} \omega_{i,j})$
- $x_{a,l,c} = \begin{cases} 1 & \text{si la case (l,c) est attribuée à a} \\ 0 & \text{sinon.} \end{cases}$

Contraintes:

- $\sum_{l=0}^{h-H(a)-1} \sum_{c=0}^{w-W(a)-1} y_{a,l,c} \leq 1, \forall a \in A$
- $\sum_{a \in A} x_{a,l,c} \leq 1, \forall \begin{cases} l \in [\![0; h-1]\!] \\ c \in [\![0; w-1]\!] \end{cases}$
- $y_{a,l,c} \leq x_{a,l+i,c+j}, \forall \begin{cases} l \in [\![0; h-H(a)-1]\!] \\ c \in [\![0; w-W(a)-1]\!] \\ i \in [\![0; H(a)-1]\!] \\ j \in [\![0; W(a)-1]\!] \end{cases}$
- $a_{a,l,c} \leq \sum_{i=0}^{min(l;H(a))} \sum_{j=0}^{min(c;W(a))} y_{a,l-i,c-j}, \forall \begin{cases} a \in A \\ l \in [\![0; h-1]\!] \\ c \in [\![0; w-1]\!] \end{cases}$

La fonction objectif cherche à maximiser les gains pour tous les annonceurs et toutes les cases on fait la somme des gains si l'annonceur est sur la case.

La variable w indique le prix payé par l'annonceur a s'il est placé en case (l,c). Soit la somme des prix des cases utilisées, soit le max qu'il est prêt à payer.

La première contrainte vérifie qu'un annonceur ne soit placé qu'une seule fois sur la grille.
La deuxième contrainte permet de vérifier qu'il y ai au plus un annonceur par case.
La troisième et la quatrième contrainte permettent de réserver des cases et de vérifier qu'il n'y ai pas de chevauchement entre les annonceurs.

Avec GLPK et sur les tests unitaire fournis, nous avons les temps d'exécution suivants:

| tests unitaires | w*h | n | temps (ms) |
|---|---|---|---|
| test U1 | 25 | 3 | 9.87 |
| test U2 | 25 | 3 | 17.80 |
| test U3 | 25 | 3 | 11.26 |
| test U4 | 25 | 1 | 2.89 |
| test U5 | 4 | 10 | 2.19 |
| test U6 | 100 | 10 | 57.09 |
| test U7 | 4 | 5 | 1.04 |
| test U8 | 9 | 2 | 1.88 |
| test U9 | 5 | 3 | 1.96 |
| test U10 | 5 | 3 | 2.01 |

## Evaluation des performances :

La mesure des tests s'est fait sur le temps d'exécution pour trouver un résultat.
Nous avons tenté de réaliser des tests sur de plus grandes instances que les tests unitaires (grille de 1000x1000 et 400 annonceurs), mais la mémoire de la machine finissait toujours par être surchargée. Nous avons donc dû nous contenter des résultats des tests unitaires présentés ci-dessus.

Machine utilisée:

- Processeur: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz
- RAM: 8Go - DDR3

Solveur utilisé:

- GLPK

Chaque jeu de test a été lancé 10 fois et les valeurs extraites sont celles des moyennes de chaque test.

Sans surprise, les tests ayant les meilleurs résultats sont ceux qui limitent le nombre de possibilités. Les tests avec un très grand nombre de possibilités ont des performances qui se dégradent très vite.

En revanche, les valeurs trouvées correspondent à la meilleure solution possible. On a donc un solveur assez long, mais avec un très bon résultat.

## Conclusion :

On peut conclure que le programme linéaire est bien plus rapide que l'algorithme exacte, en effet il ne parcourt pas toute les potentielles solutions contrairement à l'algorithme exact. Nous allons maintenant contacter le propriétaire de la lune pour lui vendre notre programme linéaire.