Optimisation de l'algorithme de force brute

RICHARD Balthazar

Dans le précédent algorithme de force brute servant à la résolution des grilles du type 4×4, nous avons oublié un paramètre permettant une importante optimisation.

En effet, pour résoudre ces grilles, nous devons chercher au maximum l'ensemble des parties d'un ensemble à n éléments, soit $card\,(\mathscr{P}(E)) = 2^n$. Plus exactement, nous devons tester $2^n - 1$ combinaisons dans le pire des cas car nous ne commençons pas à 0 mais à 1 appui (on ne prend pas en compte l'ensemble vide $\emptyset$).

Dans notre cas, nous avons donc $2^{16} - 1 = 65535$ combinaisons au maximum.

L'algorithme que nous avions conçu simulait donc bien des appuis.
Le problème est le suivant : à chaque itération i, il renvoyait une liste de $\frac{16!}{(16-i)!}$ matrices représentant chacune la grille de départ affectée par i appuis.
Au maximum, le programme renvoyait donc $\frac{16!}{(16-16)!} = 16!$ grilles, ce qui ne représente donc absolument pas la somme des parties d'un ensemble à 16 éléments (même exempte de l'ensemble vide $\emptyset$).

La raison de ce problème est le "critère d'invalidité" des appuis que nous avions mis en place. En effet, nous avions bloqué dans l'algorithme la possibilité d'appuyer deux fois sur la même case lors d'une simulation. Ainsi, on ne risquait pas d'annuler un appui car, on le rappelle, deux appuis sur une même case reviennent à ne pas l'avoir touchée.

Or une autre règle du Lights out stipule que l'ordre des cases touchées n'importe pas. Le programme a donc laissé place à des grilles similaires.

Prenons cette grille pour exemple :

0	0	0	0
0	1	0	0
1	1	1	0
0	1	0	0

Mettons en place la convention suivante :

— la case en rouge est la première touchée et la case en vert est la seconde ;

— Les appuis sur les cases x et y peuvent s'écrire comme le "parcours" $[x\ ,\ y]$

Deux appuis peuvent donner les grilles suivantes :

x	0	0	0
0	x	1	0
1	0	1	0
0	1	0	0

A gauche, on a effectué un appui sur les cases [1 , 6].

x	0	0	0
0	x	1	0
1	0	1	0
0	1	0	0

A droite on a effectué un appui sur les cases [6 , 1].

On a donc deux grille similaires, or dans la notions de parties d'un ensemble, il n'y a pas la notion d'ordre. Une grille générée par les appuis sur les cases x suivi de y est la même qu'une grille générée par les appuis sur les cases y suivis de x [1].

Pour optimiser le code, il faut enlever les cas de grilles similaires !
Nous allons, pour ce faire, modifier le "critère d'invalidité". Il faut appuyer sur une case si et seulement si son numéro est strictement plus grand que celui de la précédente. Ainsi, si nous avons déjà une grille générée par les appuis [1 , 2], nous n'aurons pas la grille générée par [2 , 1], qui est exactement la même.
Comme cela, nous n'aurons plus de cases "doublons".

Ancienne version, non optimisée, du programme :

```
for case in range(longueur):
        # On vérifie que les cases n'ont pas déjà été appuyées
        # (sinon cela annulerait l'appui de la case).
        if parcours[case][0] == lignes and parcours[case][1] == colonnes :
                valeur = False
```

Version optimisée du programme :

```
for case in range(longueur):
        # On vérifie que les cases n'ont pas déjà été appuyées
        # et que les cases sur lesquelles on veut appuyer sont
        # strictement plus grandes que les précédentes.
        if lignes < parcours[case][0] or \
        lignes <= parcours[case][0] and colonnes <= parcours[case][1] :
                valeur = False
```

C'est la variable ***valeur*** qui sert à discriminer des cases en empêchant la simulation d'appuis.

1. Dans le cas de deux appuis bien sûr (avec $0 \leq x, y \leq 16$ donc).

Ainsi à chaque itération i, le programme optimisé renvoie $\binom{16}{i}$ grilles représentant i appuis simulés sur la grille de départ, ce qui est bien moindre que $\frac{16!}{(16-i)!}$ [2].

Au maximum, cela représente donc bien l'ensemble des parties d'un ensemble à 16 éléments (toujours sans $\emptyset$ bien évidemment), contrairement au précédent algorithme.

Enfin, pour évaluer l'optimisation, comparons les temps d'exécution des deux versions :

Considérons la grille de départ :

1	1	0	1
0	0	0	1
0	1	0	0
1	1	1	1

Celle-ci est donc solvable par l'appui de 5 cases au minimum.

```
METHODE 1 :

La solution est : [[0, 1], [0, 3], [2, 1], [3, 0], [3, 2]]
Cela a mis 15.266294002532959 secondes.


METHODE 2 :

La solution est : [[0, 1], [0, 3], [2, 1], [3, 0], [3, 2]]
Cela a mis 0.32067179679870605 secondes.
```

La méthode optimisée est, dans ce cas précis, près de 47.6 fois plus rapide !

2. $\binom{16}{i} = \frac{16!}{i! \times (16-i)!}$ et $\frac{16!}{i! \times (16-i)!} \leq \frac{16!}{(16-i)!}$