PTS I

Exploration Exhaustive du Redi Cube

26 novembre 2020

Étudiants :

Gabriel	VANNIER	gabriel.vannier@edu.devinci.fr
Ulysse	BERTHET	ulysse.berthet@edu.devinci.fr
Francis	KULUMBA	francis.kulumba@edu.devinci.fr

Tuteurs :

Jean-Philippe	LELIEVRE
Sophie	DEPEYRE

Mots-Clés : Redi Cube; Game Theory; Solver

Abstract :

Il existe de nombreuses variantes du Rubik's Cube, dont la complexité peut être évaluée grâce à une exploration exhaustive de tous les mélanges possibles et à la manière de passer de l'un à l'autre. Un bon exemple est le Rubik's Cube, pour lequel on sait que le "Nombre de Dieu" est de 20, ce qui signifie que tout mélange peut être résolu par une combinaison de 20 mouvements maximum. L'objectif de ce projet est de mener une étude similaire sur une variante récente du Rubik's Cube, le Redi Cube, dont le nombre de Dieu n'est pas encore connu. En plus de l'aspect "exploration exhaustive", on peut s'intéresser à des aspects tels que la recherche de méthodes pour résoudre le Redi Cube et la création d'une interface graphique pour faciliter l'utilisation.

Table des matières

Introduction

Le Redi Cube est une variante du très populaire Rubik's Cube. A la différence de son cousin, les mouvements sur le Redi Cube s'appliquent sur les sommets. Notre objectif est de trouver un algorithme pour le résoudre de manière optimale (en effectuant le minimum de mouvements nécessaires).

L'ambition du projet est de rendre au Puzzle Club France un algorithme capable de proposer une solution pour tous les mélanges du cube. Cet algorithme, construit à l'aide d'un cheminement mathématiques ou d'un modèle de learning, se doit de répondre à certains critères de performances :

- Résoudre n'importe quel mélange en moins de 1 minutes.
- Tourner sur un ordinateur portable "standard" (8Go de Ram, processeur i5 7ème génération...).
- Le nombre de mouvements que l'algorithme effectue pour résoudre un mélange doit être inférieur ou égal au nombre de mouvements qu'on a effectuer pour mélanger le cube.

Si nous remplissons dans les délais tous les objectifs nécessaire du projet nous nous attellerons à des objectifs secondaires et optionnels :

- Avoir un algorithme suffisamment performant pour faire un exploration exhaustive de toutes les positions du redi cube (ce qui implique de résoudre environ 5 millions de positions par seconde).
- Trouver une estimation du nombre de Dieu du Redi Cube (nombre maximal de mouvements nécessaires pour résoudre n'importe quel mélange). Cette estimation peut être deux bornes le plus rapprochées possible ou, au mieux, une valeur exacte.
- (uniquement si nous réussissons à trouver une estimation du nombre de Dieu) nous pourrons soumettre une démonstration qui prouve que notre approche est correcte et exacte. Solution trouvée soit par exploration exhaustive ou seulement sur un échantillon de la population.
- Proposer une modélisation visuelle du Redi Cube, en 3 dimensions et animé permettant de voir la résolution du cube en temp réel

L'algorithme doit être livré, fonctionnel, au plus tard le 1er Avril 2020. Le Puzzle Club France étant une association caritative, nous réalisons ce projet a bénévolement avec un budget total de 0€

1 Modélisation du Redi Cube

Dans cette première partie nous discutons d'une façon simple de modéliser le Redi Cube sur papier mais également quelle structure de données est la plus adaptée dans une logique de réduction de complexité sans perdre en lisibilité du code.

1.1 Modélisation du Rubik's Cube

Une publication traitant d'un moyen de trouver une borne supérieur pour le Rubik's Cube [1] à l'aide d'un solveur, utilisant l'algorithme de Kociemba nous a fournis une première approche quant à la manière de modéliser le cube. Chaque face est nommée en fonction de l'orientation qu'elle fait face (F, U, D, B, L et R signifiant respectivement Front, Up, Down, Back, Left et Right) ; un ensemble de mouvements S a également été déterminé : chaque face tourne autour de son axe normal (il y a 6 axes normaux) dans le sens des aiguilles d'une montre sont dans le sens inverse. Ainsi, écrire R signifie que nous faisons tourner la face droite dans le sens des aiguilles d'une montre et D' signifie que nous faisons tourner la face inférieure dans le sens inverse des aiguilles d'une montre (également écrit comme D2 ou Di selon la source).

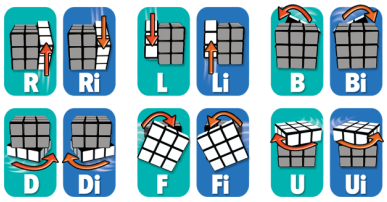

FIGURE 1 – Ensemble de mouvements du Rubik's Cube rubiks.com

1.2 Solution retenue

Le Redi Cube, contrairement au Rubik's Cube, tourne autour de ses sommets. Ainsi, le jeu de mouvements est légèrement différent de celui du Rubik's Cube et le modèle de dénomination n'est pas pertinent. Après reflexion, nous sommes arrivés à ces quelques règles :

— Le référentiel est la face du cube dirigé directement vers l'observateur.
— Un mouvement est défini par un tier tour (i.e. 120°) autour d'un sommet de cube.
— Le Redi Cube est dans un état à partir du moment où tous ses mouvements sont accomplis.

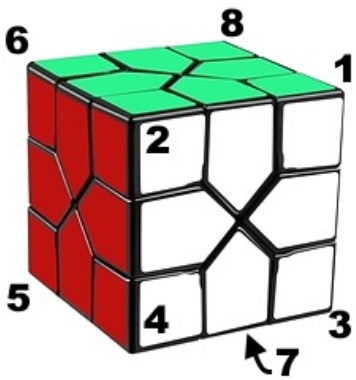

FIGURE 2 – Notation du Redi Cube où le référentiel est la face blanche

— Un mouvement peut être dans le sens des aiguilles d'une montre ou dans le sens inverse. Une double rotation est inutile car elle équivaut à faire un mouvement dans l'autre sens.
— Un Redi Cube a six faces, mais un mouvement en implique toujours trois. Nous devons donc nommer les sommets de manière à ne pas créer de confusion. Chaque sommet est numéroté de 1 à 8 en précisant quelle face du cube fait office de referentiel.
— Le Redi Cube a un ensemble de 16 coups possibles à chaque état du cube. Cet ensemble de coups est appelé S. De telle sorte que **1** est une rotation dans le sens des aiguilles d'une montre du coin supérieur droit du cube, tout référentiel confondu.
— La distance est le nombre de coups nécessaires pour atteindre un autre état

1.3 Structure de données

Le choix a été fait de programmer un solver et un explorer d'état sur Python. Des tests on été effectué à l'aide du module timeit [1] et la structure de données la plus adaptée est une matrice [2] de $6 * 3 * 3$ ou la première dimension représente la face, la deuxième, la ligne et la dernière, le carré de couleur. Cette structure de données permet d'effectuer rapidement des comparaisons mais également de coder simplement un mouvement.

```
mat = [
[[i,i,i],[i,'X',i],[i,i,i]],
[[i,i,i],[i,'X',i],[i,i,i]],
[[i,i,i],[i,'X',i],[i,i,i]],
[[i,i,i],[i,'X',i],[i,i,i]],
[[i,i,i],[i,'X',i],[i,i,i]],
[[i,i,i],[i,'X',i],[i,i,i]],
      ]
```

Les couleurs i sont représentées par les lettres : 'R','W','O','Y','G','B'. La case du centre, vide est marquée par "X".

D'autres solutions ne sont pas écartés comme un dictionnaire [3] de liste où chaque clé représente une face. Les clés sont haché sous Python et ainsi, l'accès à une clé se fait plus rapidement que l'accès à un index.

1. https ://docs.python.org/fr/3/library/timeit.html
2. https ://docs.python.org/fr/3/tutorial/datastructures.html
3. https ://docs.python.org/fr/3/tutorial/datastructures.htmldictionaries

2 Exploration et Algorithmes

Dans cette partie, des propriétés plus ou moins triviales du Redi Cube sont explorées, du nombre d'agencement possible au possible sous-ensembles. Un premier algorithme de résolution, réalisable à la main sera proposé ainsi que son implémentation en pseudo-code.

2.1 Combinaisons

Le Redi Cube cube est composé de 8 sommets et de 12 arêtes.

— Les 12 arêtes peuvent chacunes s'orienter dans deux directions. La direction de la dernière arête est fixée par la direction des arêtes précédentes, cela nous donne donc 2^{11} possibilités d'orientations des arêtes.

— Les 8 coins peuvent chacun s'orienter dans trois directions. Cela nous donne donc 3^7 possibilités d'orientations des coins.

— Les 12 arêtes peuvent se répartir dans 12 emplacements. Cela nous donne donc $12\,!$ possibilités de placement des arêtes.

On arrive donc à un total de :

$C = 2^{12} * 3^8 * 12!$

$C = 12\ 872\ 620\ 022\ 169\ 600$

$C \approx 1,29.10^{16}$

Soit un résultat plus faible que les $4,3.10^{19}$ combinaisons du Rubik's Cube mais néanmoins non-négligeable et impossible à garder en mémoire avec les moyens académiques que nous possédons.

2.2 Algorithme de résolution et sous-ensemble

A l'instar de l'algorithme Two-Phase de Herbert Kociemba[2], il existe un algorithme de résolution du Redi Cube[3] exploitant le même principe de résolution en deux phases.

1. Résoudre 4 sommets sur la première couche.
2. Résoudre la première couche.
3. Résoudre les 4 sommets restants.
4. Résoudre les 4 arrêtes de la couche du milieu.
5. Permutation des arrêtes sur la dernière couche.

Si l'on définit la première phase de l'algorithme de résolution comme étant les deux premières étapes, nous arrivons à séparer le problème en deux sous-problème dont le deuxième comporte un sous ensemble de combinaisons C' du cube beaucoup plus faible. Le Redi Cube ne sera plus composé que de 8 arrêtes et 4 sommets :

— Les 8 arêtes peuvent chacunes s'orienter dans deux directions. La direction de la dernière arête est fixée par la direction des arêtes précédentes, cela nous donne donc 2^7 possibilités d'orientations des arêtes.

— Les 4 sommets peuvent chacun s'orienter dans trois directions. Cela nous donne donc 3^4 possibilités d'orientations des sommets.

— Les 12 arêtes peuvent se répartir dans 8 emplacements. Cela nous donne donc $8\,!$ possibilités de placement des arêtes.

Soit un total de 418 037 760 combinaisons possible, ce qui est un nombre beaucoup plus raisonnable afin de pouvoir procéder à une exploration numérique du cube en Python sachant qu'une simple liste peut enregistrer jusqu'à 536 870 912 éléments sur un système 32 bits [4]. Ce premier algorithme nous permet de réaliser les deux premières étapes avec au plus 32 (?) coups au minimum 0. Quand à la seconde étape, plusieurs méthodes s'offrent à nous : [finir de rédiger]

4. https ://tinyurl.com/yx9mu66f

2.3 Pseudo-code de la première phase

Algorithm 1: step1(list)

```
initialisation;
j ← 0;
white ← list[0];
for count ← 1 ; count ≤ 2 ; count += 2 do
    while white[count][j] != "W" do
        if "W" in list[0 then
            r
        else
            o
        end
        tate(list, 0);
    end
    j = (j + 2)%4;
end
```

Algorithm 2: step2(list)

```
initialisation;
locate the next withe edge;
for count ← 1 ; count ≤ 2 ; count += 2 do
    while white[count][j] != "W" do
        rotate(list, 0);
    end
    j = (j + 2)%4;
end
```

3 Résultats

Conclusion

Références

[1] R. Tomas, “Twenty-five moves suffice for rubik’s cube,” p. 10, Mar. 2008. [Online]. Available : https://arxiv.org/abs/0803.3435

[2] K. Herbert, “Two-phase algorithm,” 1991. [Online]. Available : http://kociemba.org/math/imptwophase.htm

[3] C. Jon, “Redi cube solution,” Jan. 2020. [Online]. Available : https://jonchammer.com/2020/01/05/redi-cube-solution/