Utilisation de techniques de Machine Learning pour résoudre un Rubik's Cube

MFSN - A25
Raphaël Le Lain

Sommaire

Introduction

Le Rubik’s Cube est un Puzzle inventé par Ernő Rubik, un sculpteur et professeur d'architecture hongrois, qui s'intéresse à la géométrie et à l'étude des formes en 3D. Il reste aujourd’hui toujours très populaire, notamment dû à une présence forte d’une scène compétitive, dans laquelle les participants tentent de résoudre le Rubik’s Cube le plus rapidement possible (speedcubing). Aujourd'hui, les meilleurs scores des humains tournent autour des 3 à 4 secondes seulement. C’est sans doute le casse tête le plus populaire de l’histoire, s’étant vendu à plus de 500 millions d'exemplaires originaux et dérivés.

Le jeu, dans la version à laquelle nous allons nous intéresser dans ce rapport, consiste à manipuler un cube de 3 x 3 x 3 sous cubes via des rotations de lignes et de colonnes afin d’obtenir une unique couleur sur chaque face du cube.

À l’instar des échecs, c’est aujourd’hui un autre de ces jeux dans lequel les meilleurs humains ne peuvent plus rivaliser avec les meilleures machines. En effet, en mai 2024, l’entreprise Mitsubishi Electric battait le record du monde avec sa machine le résolvant en 0,305 secondes. Ce robot et la plupart de ses concurrents utilisent une méthode hybride : des méthodes de Machine Learning pour reconnaître les couleurs du cube, et un algorithme découvert et écrit à la main par des humains pour la stratégie de résolution : l’algorithme de Kociemba (ou du moins une variante très optimisée). Cet algorithme permet de résoudre le Puzzle presque toujours entre 18 et 22 mouvements, là où les humains utilisent généralement entre 50 et 60 mouvements pour résoudre le cube (en compétition de vitesse pure seulement, pour la compétition en mode “Fewest Moves”, les nombres de mouvements plus bas en moyenne se trouvent entre 18 et 24 coups).

En réalité, aujourd'hui, il existe des algorithmes permettant de trouver à coup sûr la combinaison de coups la plus courte possible pour résoudre un Rubik’s Cube, avec en moyenne 17.88 mouvements, comme l'algorithme IDA* de Korf. Cependant, ces algorithmes sont très gourmands en calculs, même sur un PC puissant, calculer un seul mouvement peut prendre de quelques secondes à plusieurs minutes, c’est pour cette raison que ce n’est pas la méthode utilisée par Mitsubishi Electric et ses concurrents.

Dans ce rapport, nous nous intéresserons à la question suivante : peut-on utiliser des méthodes de Machine Learning afin de trouver des stratégies qui résolvent le cube en très peu de coups en moyenne, mais dont le temps d’inférence est inférieur à celui des algorithmes optimaux ?

Nous commencerons par étudier la littérature sur le sujet, puis nous essaierons de proposer notre propre solution pour répondre à ce problème, avant de conclure.

Voies déjà explorées

I - Finding Optimal Solutions to Rubik's Cube Using Pattern Databases par Richard E. Korf (1997)

Avant ce papier, trouver une solution optimale (nombre minimum de mouvements) pour un Rubik's Cube arbitraire était considéré comme computationnellement infaisable. Les algorithmes existants (Thistlethwaite, Kociemba) trouvaient des solutions courtes mais pas nécessairement optimales.

Même si cet article n'utilise pas de techniques de Machine Learning, c'est un papier fondateur car il représente l'approche "Exacte" du problème du Rubik's Cube.

L'idée générale est de décomposer le problème : on calcule à l'avance le nombre de coups nécéssaires pour résoudre uniquement les coins ou les arêtes, et on stocke les résultats dans d'immenses tables en mémoire, qui sont ensuite utilisées pour guider la recherche, en explorant en priorité les mouvements qui rapprochent les coins de leur position finale.

Cet algorithme nous servira d'étalon pour notre solution en terme de vitesse d'inférence : si notre solution n'est pas significativement plus rapide que lui, alors elle ne présente pas d'avantages puisque cet algorithme est optimal en termes de nombre de coups. L

II - Le "Nombre de Dieu" (God's Number) (2010)

Cet article est également fondateur pour notre sujet, car il a montré que toutes les configurations possibles du cube peuvent être résolues en 20 coups ou moins.

Il a utilisé la force brute, en calculant toutes les possibilités, mais astucieusement en exploitant les symétries du cube et en partitionnant les différentes configurations en "cosets", puis en utilisant l'algorithme de Kociemba sur les 2.2 milliards de sous-ensembles obtenus pour borner le nombre de coups nécessaires pour résoudre le Rubik's Cube.

En utilisant l'algorithme IDA* de Korf, les chercheurs ont ensuite pu montrer par échantillonnage que la moyenne empirique de longueur de la solution optimale est d'environ 17.88 coups. Ce chiffre nous servira d'objectif à atteindre pour notre algorithme de Machine Learning : si la longueur moyenne des solutions que l'on trouvera se trouvent proche de 17.88, cela signifie qu'elles sont quasiment systématiquement optimales.

III - Solving the Rubik's Cube with Deep Reinforcement Learning and Search (DeepCubeA) (2019)

Des tentatives d'utiliser des techniques de Machine Learning pour résoudre le Rubik's Cube existaient déjà avant cet article, mais elles n'ont jamais donné de résultats satisfaisants : elles restaient parfois bloquées ou prenaient plus de 100 mouvements pour résoudre un cube qui n'en demandait que 20.

Cet article est une vraie référence car il a un taux de succès de 100%, c'est à dire que l'algorithme ne se bloque jamais, et propose des solutions de longueurs très correctes, entre 21 et 28 coups pour la quasi intégralité des positions possibles. Il réussir à accomplir cela sans aucune connaissance humaine injectée.

Pour accomplir cela, les chercheurs partent du Cube résolu, et appliquent un nombre de mouvements aléatoires. On entraîne ensuite le modèle à retrouver l'état résolu en utilisant le Reinforcement Learning.

Tous les articles suivants sur le même sujet se comparent à celui-ci en termes de performance, c'est sans doute la plus grande référence dans ce domaine.

IV - CubeTR: Learning to Solve the Rubik's Cube Using Transformers (2021)

Ce papier propose une approche très novatrice en traitant la résolution du cube comme un problème de traitement du langage naturel (NLP).

Cela permet de mettre à profit l'architecture Transformer, très populaire et performante, utilisée dans des systèmes comme les modèles de ChatGPT. On traite l'état du cube comme une longue séquence de tokens, dans laquelle chaque facette du cube est considérée comme un mot dans une phrase. On entraîne ensuite le modèle à traduire un état mélangé du cube en une séquence de mouvements. On utilisant un mécanisme d'attention parcimonieuse, le modèle peut se concentrer uniquement sur les parties pertinentes de la séquence.

Il apprend en observant des millions de paires (état mélangé, résolution), qui sont générées par l'algorithme de Kociemba. Le modèle obtenu n'est donc pas optimal, mais il permet de résoudre des grands lots de cubes de parallèlement assez rapidement.

V - Résolution du Rubik's Cube par Apprentissage par Renforcement List3n (2025)

Enfin, cet article réalisé par des chercheurs de l'UTT propose sa propre version de la résolution du Rubik's Cube par Apprentissage par Renforcement. L'algorithme de recherche utilise Monte Carlo Tree Search pour explorer l'arbre de manière asymétrique en équilibrant l'exploration et l'exploitation. Après une tentative d'utilisation d'ADI (Le réseau apprend à estimer le "coût à venir" (distance) en partant de l'état résolu et en remontant par des mélanges aléatoires), qui menait à des problèmes de convergence, la méthode finale utilise le Deep-Q learning (on cherche à maximiser la qualité d'une action Q dans une situation s, c'est un concept à la base du Reinforcement Learning), avec des mécanismes comme le replay buffer (pour briser la corrélation temporelle) et un réseau cible (pour stabiliser le signal d'apprentissage).

Le modèle ainsi obtenu arrive à résoudre des Rubik's Cube mélangés jusqu'à 6 fois, mais le manque de moyens de l'équipe n'a pas permis de résoudre systématiquement des états aléatoires de façon quasi-optimales.

Ma tentative

I - Représentation du Rubik’s Cube.

Le premier défi rencontré était de développer une implémentation pratique pour notre Rubik’s Cube. L’idée initiale consistait à ne pas représenter toutes les facettes de toutes les faces pour éviter les redondances. J’ai donc tenté d'opter pour une représentation qui stockait la position et orientation des différentes pièces mobiles (coins et arêtes). Cependant, cela compliquait trop les opérations à effectuer pour reproduire les différents mouvements possibles sur le cube. J’ai donc finalement opté pour un système de stockage de la couleur de chaque facette de chaque face, sauf la facette centrale, qui est déduite par l’identifiant de la face (U L F R B D pour Up Left Front Right Back Down). Cela nous amène à utiliser 8*6 = 48 variables différentes pour représenter le cube.

Dans un second temps, il a fallu implémenter les méthodes permettant les différents mouvements. On décompte un total de 12 actions possibles : deux pour chaque face, dans le sens horaire ou anti-horaire. J’ai donc implémenté la méthode rotate_face, qui prend en paramètre la face à tourner et un booléen “clockwise” pour le sens de rotation. La méthode rotate_face applique une rotation en deux étapes. D’abord, elle tourne les 8 stickers de la face concernée : comme on stocke un anneau de 8 positions, un quart de tour (90°) correspond à un décalage circulaire de 2 indices (horaire ou anti-horaire). Ensuite, elle met à jour les faces adjacentes : pour chaque face, un tableau (adjacent_map) décrit les 4 bandes de 3 stickers à déplacer. On copie ces 4 groupes, on les permute en cycle (A→B→C→D→A, ou l’inverse), puis on réécrit les valeurs pour éviter tout écrasement.

II - Génération des données et entraînement de modèles

A - Données Utilisées

Pour ce qui est des données d'entraînement possibles, j’ai identifié trois possibilités :

1 - Générer des séquences de résolutions avec Kuciemba
2 - Génerer des séquences de résolutions avec IDA* de Korf
3 - Génerer des séquences de résolutions en utilisant des scrambles (mélanges) du cube résolu.

La première option est celle utilisée dans l’article de CubeTR. Elle permet de générer des séquences assez rapidement, mais elle présente deux inconvénients. Le premier est que certaines de ses résolutions sont non-optimales, et le second et que cela retire l’avantage conceptuel de résoudre le Puzzle sans connaissance humaine.Utiliser IDA* de Korf résoudrait le premier problème mais pas le second, et ne peut pas être utilisé pour générer un dataset de taille suffisamment conséquente avec peu de puissance de calcul en des

temps raisonnables.

La troisième option est sans doute la plus rapide, et elle ne nécessite aucune connaissance humaine à proprement parler, mais c'est sans doute aussi celle qui génère des séquences de moins bonnes qualités : les séquences de résolutions seront sans doute très sous-optimales. Comme DeepCubeA obtient des résultats convaincants avec cette approche, ce sera mon point de départ.

Afin d'améliorer sensiblement la qualité des résolutions, j'ai implémenté un filtre qui retire toutes les séquences qui passent plus d'une seule fois sur un même état du cube, qui sont de manière évidente sous-optimales. Ce faisant, on retire environ 4/5ème des séquences de 20 mélanges générées (proportion augmentant avec la profondeur de mélange).

B - Approches de Machine Learning

Au cours de ce projet, l'architecture de modèle utilisée sera toujours la même : LGBM (Light-Gradient-Boosting-Machines). Il s'agit d'une variante des GBDT (Gradient-Boosted-Decision-Trees), qui offre généralement un compromis très intéressant entre le temps d'entraînement et la performance du modèle sur des grands jeux de données. Utiliser des modèles plus scalables comme des réseaux de neurones profonds serait sans-doute préférable si on disposait de plus de puissance de calcul et de temps disponible.

1. Le point de départ en termes d'approche de la modélisation n'utilise pas encore de techniques de Reinforcement Learning à proprement parler. On entraîne simplement un modèle à prédire quel mouvement précède l'état donné en paramètre (x) dans la séquence. C'est donc un problème de classification à 12 classes. Avec un dataset de 1 420 386 couples (état,mouvement), on obtient une accuracy de 0.3666 sur le test set et 0.4168 sur le train set (pas d'overfitting flagrant). Ce résultat peut sembler extrêmement faible, mais comme on se trouve dans un problème de classification à 12 classes, un mouvement aléatoire amènerait à une accuracy de 0.125. Notre modèle apprend donc bien, même si c'est dans une moindre mesure.

En essayant de résoudre de nouveaux cubes mélangés (on limite le nombre de coups à 200 pour éviter des boucles infinies), on observe qu'à 5 mélanges, tous les cubes sont résolus en 4.36 coups en moyenne, mais qu'à 10 mélanges, le taux de réussite tombe à 51% (6.98 coups en moyenne). À 20 mélanges, le taux de réussite tombe à seulement 2%.

2. La première tentative d'amélioration se rapproche plus d'une approche de Reinforcement Learning : On entraîne un premier modèle à estimer la distance en nombre de coups à l'état résolu. Cette distance nous servira de "Value" V(s), tandis que le modèle de mouvement (le modèle de la tentative précédente) nous servira de Policy pi(a|s), avec a le mouvement choisi, s l'état actuel et s' le nouvel état.

À l'inférence, on choisit l'action a qui minimise :

$$\text{score}(a) = w_{dist} * V(s') + w_{policy} * (-\log P_{pi}(a|s))$$

Avec w_dist=1.0 et w_policy=0.25, sur 420 000 couples (état, profondeur) et 400 000 couples (état, mouvement), on obtient 100% de résolutions à profondeur 5, avec 4.20 étapes en moyenne. À une profondeur de mélange de 10, on tombe à 52% de résolutions avec une longueur de 6.88 coups, et à 20 mélanges, le taux de réussite tombe à 0%.
Tel quel, il n'est pas possible d'affirmer si cela a mené à une amélioration significative des performances par rapport à l'approche précédente. Une comparaison des performances des différences entre les performances sera effectué à la fin de cette partie.

Comme on observe que le problème n'est pas tant le nombre de coups par résolution mais plutot le taux de réussite qui chute avec la profondeur de mélange, cela nous mème à une nouvelle tentative d'amélioration :

3. En supposant que les tentatives qui échouent tombent en réalité dans des boucles, on va ajouter un nouveau modèle qui va estimer pour chaque mouvement la probabilité qu'il mène à une boucle. Pour ce faire, on simule des résolutions en suivant le Policy model, et, si un même état est visité plus d'une fois, on marque tous les couples (état,mouvements) rencontrés dans cette résolution comme risqués. On entraîne donc un classifier mono-classe.
Lors de l'inférence, on minimise :
final_scores = (w_val * pred_values) + (w_pol * policy_score) + (w_loop * loop_risk * 10).

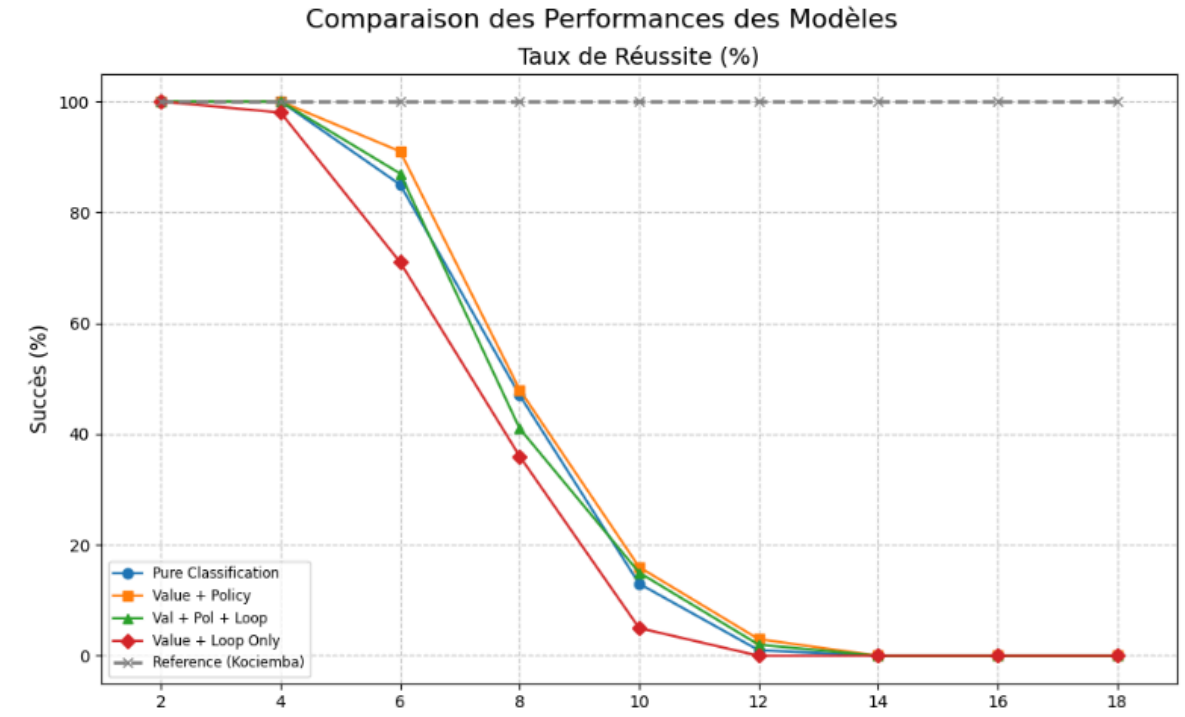
Comme précisé précédemment, on garde la comparaison détaillée pour la fin de cette partie.

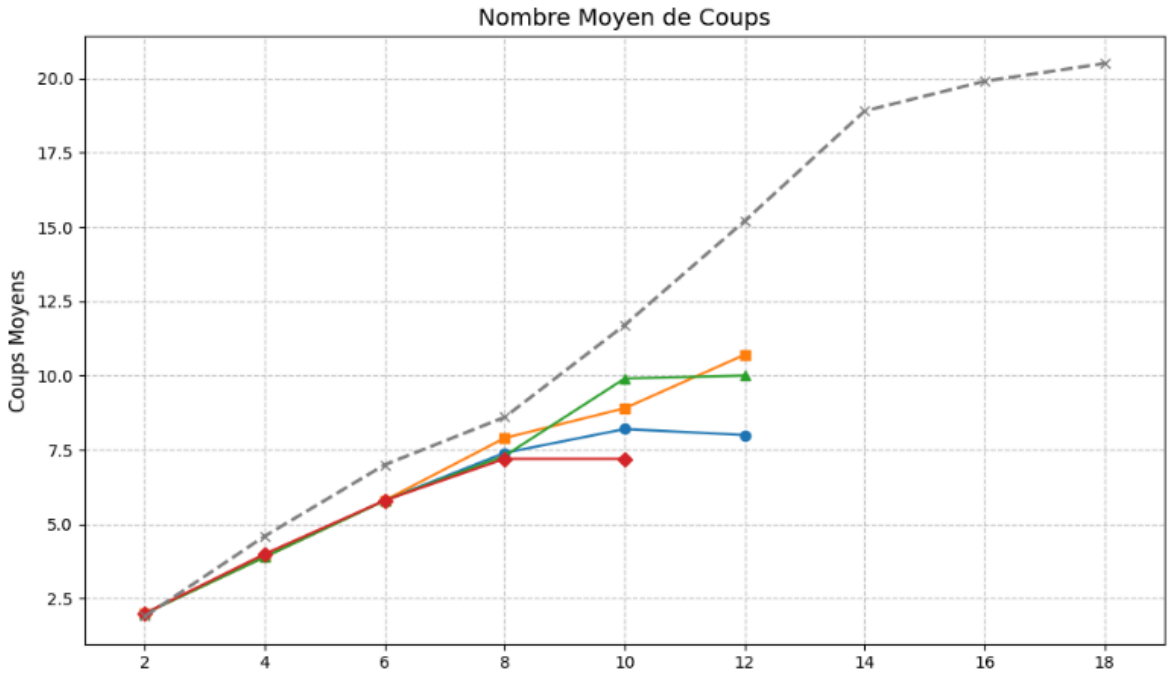
4. Une hypothèse se pose alors : est-ce que le Policy model est vraiment nécessaire ? On peut entraîner le modèle de risque de loop sur les résolutions effectuées en minimisant simplement la valeur de distance prédite. Le score à minimiser devient alors simplement :

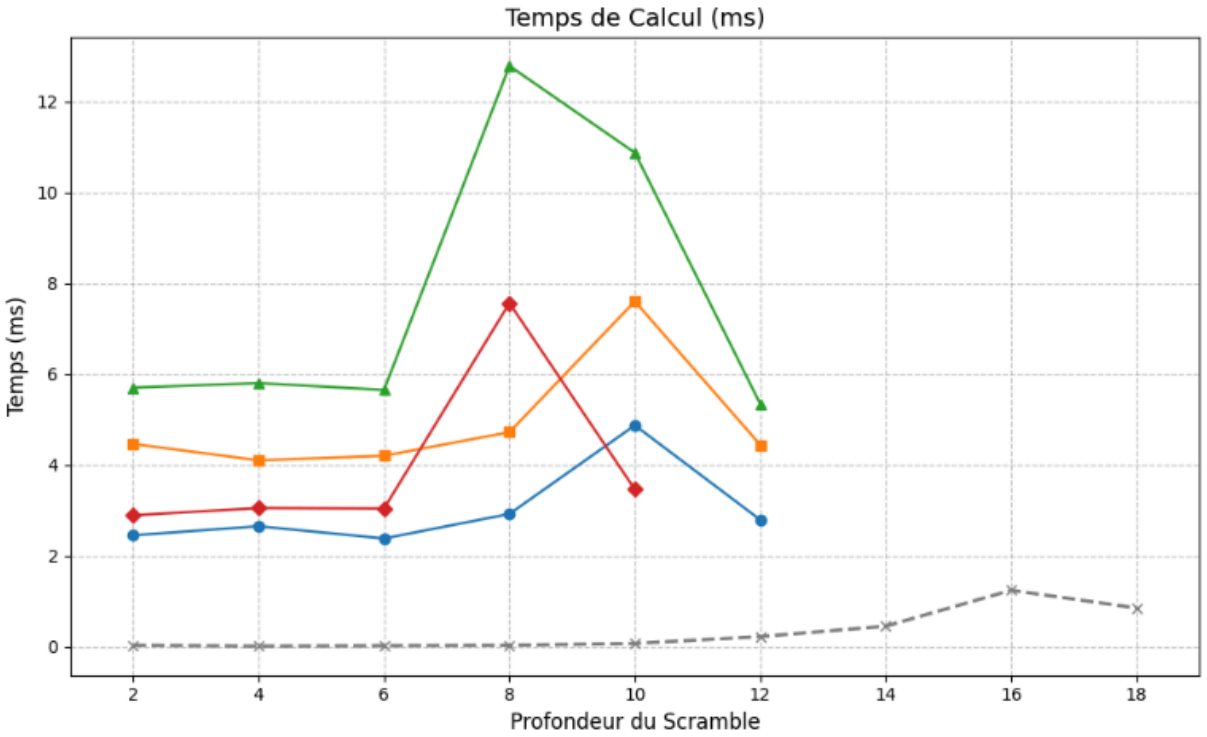
Score(m) = Distance_Estimée(État_Suivant) + (Risque_Boucle(État_Actuel, m) * Pénalité).
Cela permettrait peut-être de réduire la complexité de l'approche et le temps d'inférence sans perdre en précision

C - Comparaison finale des approches

Afin de savoir quelle approche se révèle la meilleure, et estimer la qualité du résultat, on lance toutes les approches avec les même paramètres, et on les compare à l'algorithme de Kuciemba. Avec 150 000 échantillons de test, 50 000 échantillons pour le modèle de loop, et des LGBM de 600 arbres, on obtient les résultats suivants :


Comparaison des Performances des Modèles
Taux de Réussite (%)
Succès (%)
Pure Classification
Value + Policy
Val + Pol + Loop
Value + Loop Only
Reference (Kociemba)


Nombre Moyen de Coups
Coups Moyens



Comme on peut le voir, notre modèle qui s'en sort le mieux est le modèle Value+Policy seulement, avec un taux de succès systématiquement supérieur aux autres (bien que par une marge assez faible), et un temps d'inférence par coup et un nombre de coups par résolution dans la moyenne de nos modèles. On peut tout de même s'étonner d'à quel point le modèle de classification simple est au coude à coude avec les modèles qui utilisent une approche de Reinforcement Learning.

Évidemment, aucun de nos modèles ne rivalise avec Kociemba, que ce soit en terme de taux de réussite (100% pour Kociemba systématiquement), ou en termes de temps d'inférence : environ 100 fois plus rapide que nos modèles. Pour le temps, on observe cependant que Kociemba met de plus en plus de temps à calculer les mouvements quand le cube est mieux mélangé. Peut-être que si nos modèles avaient réussi à résoudre des cubes véritablement bien mélangés (30 à 40 scrambles), ils auraient été plus rapides que Kociemba. On notera également que, pour des profondeurs de mélange très faibles, nos modèles trouvent souvent de meilleures solutions que Kociemba (à profondeur = 4, notre moyenne est à 3.9 contre 4.6 pour Kociemba. Cela peut être encourageant pour une éventuelle future mise à l'échelle.

Conclusion

En conclusion, le Rubik's Cube, de par sa nature combinatoire menant à une explosion des positions possibles, est un défi majeur pour des algorithmes de Machine Learning. Avec les techniques actuelles, à moins d'avoir accès à suffisamment de puissance de calcul, il n'est pas possible d'obtenir des modèles pouvant faire concurrence à des algorithmes classiques comme Kociemba. Comparer les modèles obtenus dans cette étude avec les modèles de DeepCubeA n'aurait également que très peu de sens au vu de l'écart significatif de moyens.

Afin de respecter les limites de temps imposées par ce projet, des sacrifices ont également dû être faits sur le temps dédié à l'entraînement. En effet, les performances obtenues par le premier modèle entrainé sur 1 420 386 échantillons était très prometteuses : 51% de résolutions de cubes mélangés 10 fois en 200 coups, contre 15% en 1500 simulations en utilisant l'algorithme MCTS guidé par un réseau de neurones dans l'article "Résolution du Rubik's Cube par Apprentissage par Renforcement ". Cependant je ne pouvais pas me permettre d'utiliser plus d'une heure sur l'entraînement de chacun de mes modèles.

Mettre à l'échelle le projet, en utilisant plus d'échantillons et éventuellement des modèles plus robustes pourrait sans doute permettre d'arriver à des performances proches de celles de DeepCube, mais sans doute toujours difficilement compétitifs en termes de temps d'inférence et de taux de réussite face à Kociemba.

Ce projet m'a tout de même permis de mieux me familiariser avec le concept d'apprentissage par renforcement, et également de mieux apprécier des problèmes du type du Rubik's Cube, dans lesquels notre donnée n'est pas polluée par du bruit, mais la quantité de possibilités met en difficulté les approches actuelles du Machine Learning. Peut-être qu'un changement de paradigme serait nécessaire pour ce type de problème pour les rendre solubles sans moyens matériels considérables.

Sources

Rokicki, T., Kociemba, H., Davidson, M., & Dethridge, J. (2014). The Diameter of the Rubik's Cube Group Is Twenty. SIAM Review, 56(4), 645-670. https://tomas.rokicki.com/rubik20.pdf

Korf, R. E. (1997). Finding Optimal Solutions to Rubik's Cube Using Pattern Databases. Proceedings of the National Conference on Artificial Intelligence (AAAI). https://www.cs.princeton.edu/courses/archive/fall06/cos402/papers/korfrubik.pdf

Agostinelli, F., McAleer, S., Shmakov, A., & Bhatti, P. (2019). Solving the Rubik's Cube with Deep Reinforcement Learning and Search. Nature Machine Intelligence, 1(8), 356-363.

Chasmai, M. E. (2021). CubeTR: Learning to Solve the Rubik's Cube Using Transformers. arXiv preprint, arXiv:2111.06036v1. https://arxiv.org/pdf/2111.06036v1

Romain Goldenchtein, Simon Gelbart, Ahmad W. Bitar. Résolution du Rubik’s Cube par Apprentissage par Renforcement. 2025. ffhal-05185396