# Rapport de projet MCTS

Karim Belhadj, Alexandre Guerassimov

**Résumé**

Les algorithmes de recherche Monte-Carlo sont typiquement utilisés pour générer des agents performants pour différents jeux et ils ont notamment gagné en popularité grâce aux performances qui ont pu être obtenues dans des jeux combinatoires populaires tels que les échecs et le jeu de go. Il est question pour ce projet de tester des algorithmes Monte-Carlo avec le jeu 2048 pour voir quels résultats on peut obtenir.

## 1 Présentation du jeu

Nous avons choisi pour ce projet de travailler sur le jeu 2048. Ce jeu est constitué d'une grille, traditionnellement de taille 4x4, avec des cases contenant des valeurs ou non. En choisissant une direction, il est alors possible de faire bouger autant que possible toutes les cases avec des valeurs dans la dite direction. Lorsque qu'une case rencontre alors une autre case de la même valeur, celles-ci fusionnent en une case d'une valeur supérieure. Le but est alors d'obtenir une case avec au moins la valeur 2048. Toutes les valeurs s'écrivent sous la forme $2^n$, avec ainsi 2048 pour $n = 11$ et lorsqu'une fusion se produit, la nouvelle case aura pour valeur $2^{p+1}$ en ayant $2^p$ pour valeur dans les cases utilisées pour la fusion. Le score se met à jour en sommant les valeurs des cases produites par des fusions.

Pour encoder un état du jeu, on se base sur le fait que toutes les valeurs possibles sont des puissances de 2 et que la valeur objectif est 2048. On souhaite donc générer une valeur distincte pour chaque état en appliquant une transformation pour changer de base. Nous avons donc utilisé la base 11 pour cible, en appliquant la transformation suivante :

$$\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \log_2 v_{ij} \times \beta^{N \times i + j}$$

Avec $N = 4$, $v_{ij}$ la valeur d'une case, et $\beta = 11$. Ainsi, il ne devrait pas y avoir de collisions jusqu'à ce que l'on obtienne des $2048 = 2^{11}$.

Une partie débute avec deux cases initialisées avec soit 2 ou 4, avec une probabilité plus grande d'avoir un 2 et après chaque action, une case aléatoire se retrouve affectée de la même façon. Si la grille est pleine sans qu'il soit possible d'effectuer une fusion, la partie est alors terminée.

# 2 Algorithmes utilisés

Nous avons testé les algorithmes UCT, RAVE et GRAVE avec des valeurs de simulations 10, 50, 100 et 600, et avec différents hyperparamètres. Avec le matériel à dispositions, seuls 5 tests par configuration ont pu être réalisés.

## 2.1 Baseline

Afin d'avoir une référence à battre, nous avons commencé par utiliser un algorithme de Monte Carlo basique où avant de jouer un coup, nous faisons un grand nombre de parties aléatoires après avoir utilisé ce mouvement. En ayant exécuté cet algorithme avec 100 simulations par coup, nous avons obtenu de bons résultats, la tuile 2048 a été atteinte.

## 2.2 UCT

L'algorithme UCT a montré de bons résultats mais ils sont moins bons que la méthode de base. Avec un UCT réalisant 100 simulations, nous n'arrivons pas à dépasser la tuile 1024. Nous nous sommes demandés pourquoi l'algorithme UCT qui est censé être une amélioration de la version basique de Monte-Carlo fonctionne moins bien. L'une des explications à laquelle nous avons pensé concerne la manière dont l'algorithme définit ce qu'est une bonne partie. Pour un état donné, on cherche à faire le mouvement qui mène à un état ayant le score le plus prometteur. Or, le nombre d'états étant très élevé, il est très probable que plusieurs "bons" états n'aient pas été visités, ou aient eu par malchance un mauvais score. Le choix du mouvement à partir des connaissances de l'algorithme est donc biaisé par les parties aléatoires ayant eu un score élevé. Même en changeant la constante UCT nous n'avons pas réussi à avoir d'aussi bons résultats que la méthode ne faisant que de l'exploration aléatoire, ce qui nous a beaucoup étonné.

Nous avons donc essayé l'algorithme UCT avec différentes valeurs de la constante, puis nous avons regardé les scores que ces variantes arrivaient à atteindre. Ci-dessous, un graphique présentant les performances de UCT en fonction du nombre de simulations et de la constante.

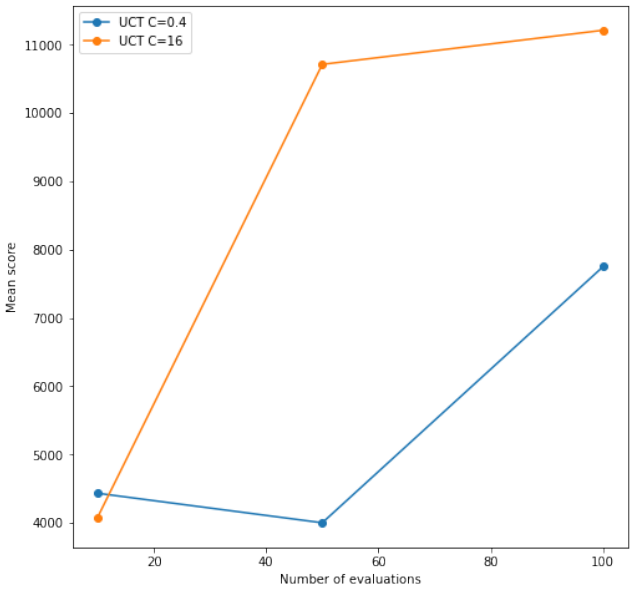


FIGURE 1 – Moyennes des scores avec différents UCT

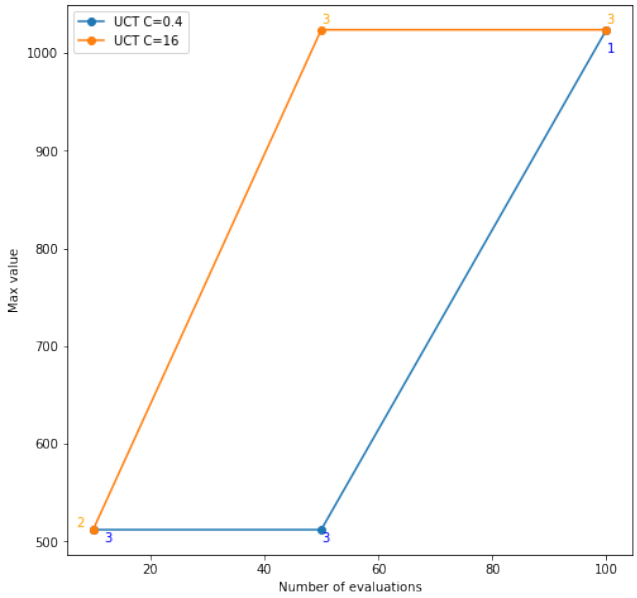


FIGURE 2 – Valeurs maximales avec différents UCT

On voit que qu'avec $c = 16$ pour UCT, l'algorithme est bien plus performant que sa version avec $c = 0.4$. Cette constante permet d'atteindre de meilleures scores et des tuiles ayant de plus grandes valeurs. On en déduit que cette valeur permet de faire un meilleur compromis entre l'exploration et l'exploitation. Des tests ont également été réalisés avec d'autres constantes, mais compte tenu du temps d'exécution, ils n'ont été réalisés que pour une unique valeur d'exploration. Il n'y a donc pas assez de valeurs pour pouvoir en faire des courbes.

Cette méthode nous permettrait de battre un humain ayant un niveau assez moyen au jeu. Comparons-le aux autres méthodes que nous avons appliquées.

## 2.3 RAVE

Nous avons également essayé d'appliquer l'algorithme RAVE et nous avons étudié l'effet du biais sur les résultats obtenus.

Le biais de $1 * 10^{-7}$ permet en moyenne d'avoir de meilleurs scores que le biais de $1 * 10^{-5}$. Cependant, il n'a pas été possible durant nos essais d'obtenir la tuile 2048 avec le biais $1 * 10^{-7}$ et elle n'a été obtenue qu'une seule fois avec l'autre biais. Mais il faut prendre en compte le faible nombre de tests réalisés, et compte tenu des scores moyens, il est possible qu'avec un peu plus de chance, la tuile 2048 aurait pu être obtenue avec un biais de $1 * 10^{-7}$. On peut donc en déduire

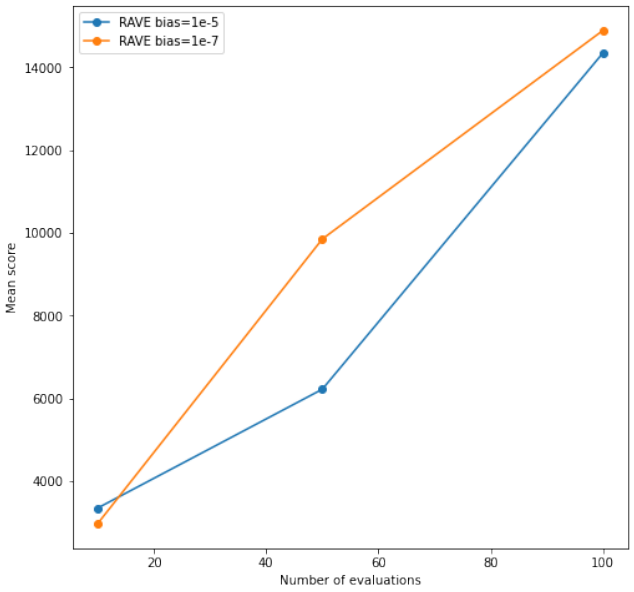


FIGURE 3 – Moyennes des scores avec différents RAVE

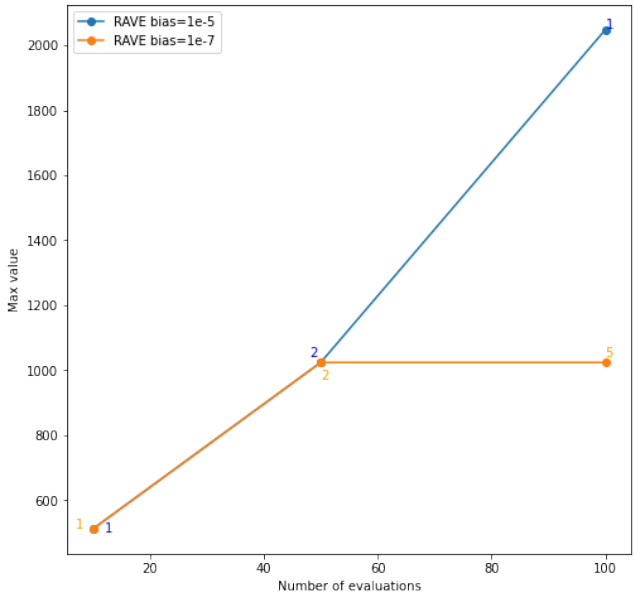


FIGURE 4 – Valeurs maximales avec différents RAVE

que dans notre cas, $1 * 10^{-7}$ est le meilleur biais pour RAVE.

### 2.4 GRAVE

Nous pouvons observer Figure 6 qu'avec l'algorithme GRAVE, nous arrivons à gagner au jeu en atteignant la valeur 2048 avec 100 évaluations par coup. On peut également remarquer Figure 5 qu'avec un biais plus bas permet d'avoir de meilleurs scores en moyenne. Malgré une faible consistance dans les valeurs maximales obtenues (avec une occurrence à chaque fois), le fait que le score reste en moyenne supérieure pour un biais à $10^{-7}$ laisse à penser qu'il s'agit sans doute là de la meilleure valeur pour le paramètre en question.

## 3 Comparaison

Après tous ces tests, nous pouvons donc comparer les résultats des différents algorithmes entre eux. Dans un premier temps, nous pouvons comparer les algorithmes utilisant les meilleurs hyper-paramètres.

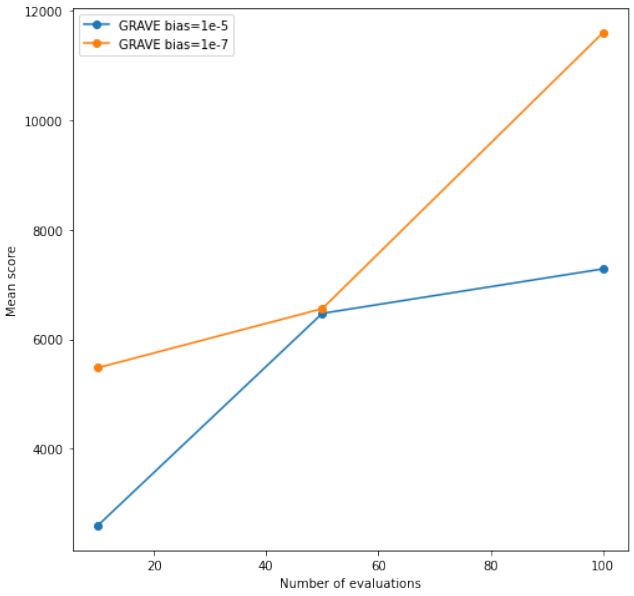


FIGURE 5 – Moyennes des scores avec différents GRAVE

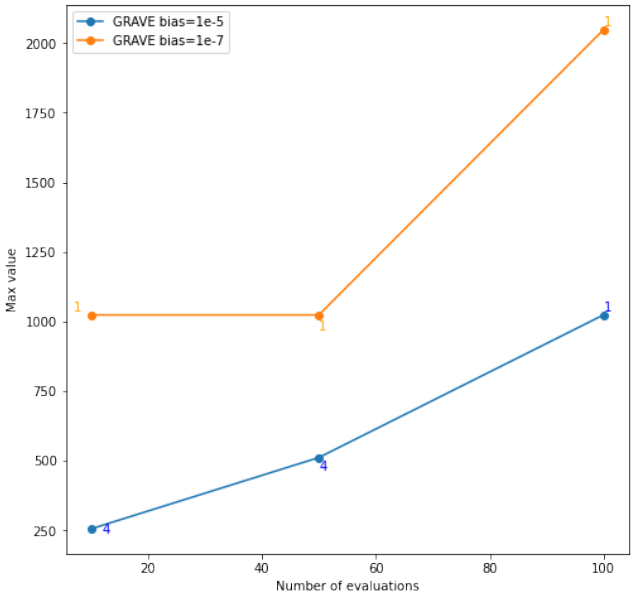


FIGURE 6 – Valeurs maximales avec différents GRAVE

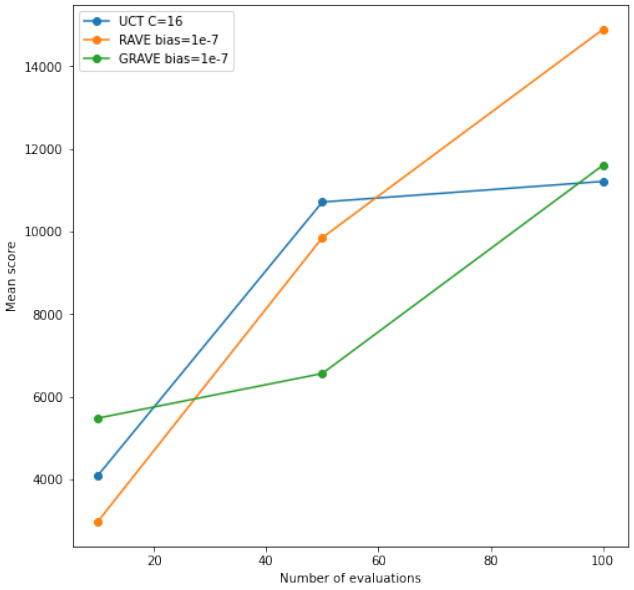


FIGURE 7 – Comparaisons scores

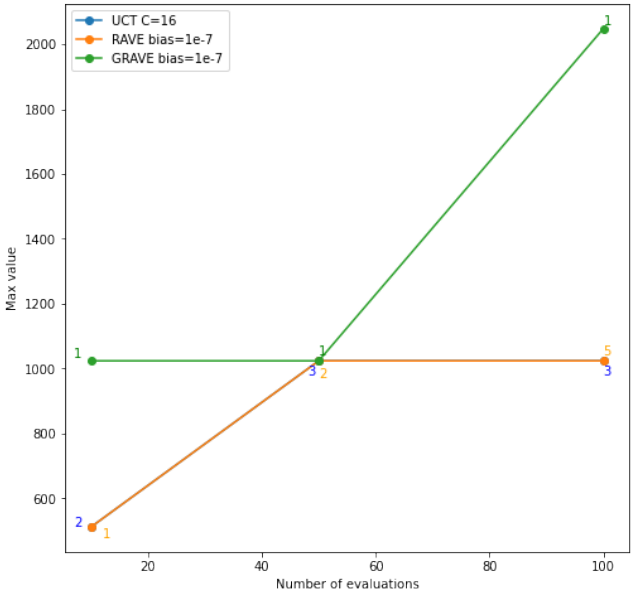


FIGURE 8 – Comparaisons valeurs

Comme le montre les courbes ci-dessus, RAVE est l'algorithme le plus performant du point de vue du score moyen pour 100 simulations. Malheureusement, la tuile 2048 n'a pas été atteinte avec cet algorithme. Nous pouvons également observer que pour 50 observations UCT arrive à de meilleurs scores moyens que RAVE et que même si GRAVE a ici des scores moyens faibles, c'est le

seul à parvenir à atteindre l'objectif. Il est difficile en l'état de pouvoir tirer une conclusion viable de ces résultats.

Dans un second temps, nous avons comparé les résultats des différents algorithmes avec les hyperparamètres les moins performants. Comme ces algorithmes sont moins performants, nous avons pu les exécuter avec un plus grand nombre de simulations, car étant donné que les parties se finissent plus vite, les temps d'exécution étaient donc moins importants pour les premières simulations. Toutefois, il ne faut pas compter moins de 3 heures pour une partie avec GRAVE en faisant 600 simulations par coup, et les autres algorithmes demandent environ le même volume horaire.

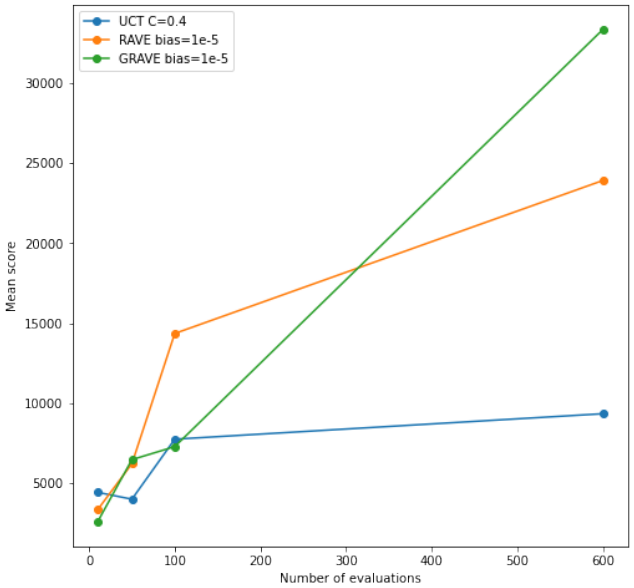


FIGURE 9 – Comparaisons scores

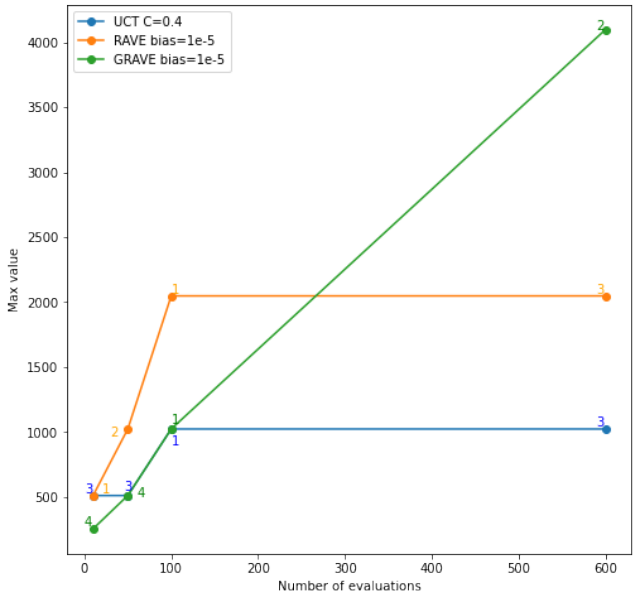


FIGURE 10 – Comparaisons valeurs

Comme il est possible de l'observer ci-dessus, GRAVE parvient à atteindre la tuile 4096 en faisant 600 simulations par coup. C'est également l'algorithme qui parvient à obtenir le plus grand score moyen pour ce même nombre de simulations.

On observe également que GRAVE ne présente pas de très bonnes performances lorsque le nombre de simulations est relativement faible. Pour 100 simulations il est équivalent à UCT en terme de performances et il est bien moins bon que RAVE.

# 4 Conclusions

En conclusion, parmi les algorithmes que nous avons appliqués, RAVE et GRAVE présentent les meilleures performances. RAVE sera meilleur si nous nous autorisons à faire un relativement petit nombre de simulations, tandis que GRAVE sera bien plus performant lorsque le nombre de

simulations est conséquent. Mais tout ceci est à prendre en compte sachant que peu de tests ont pu être réalisés avec le matériel à disposition ; ainsi, des résultats différents pourraient éventuellement être obtenus dans d'autres conditions.