Rapport Projet Blobwar

Algorithmique avancée

PITOIS Simon, NEMO Maxime

24 mars, 2022

Contents

Notations

Par la suite, nous noterons n_i le nombre d'états finaux possibles (profondeur max ou plus de mouvement possible) (avec éventuelle redondance) à partir de l'état initial depuis lequel on cherche le mouvement optimal. On notera $n = \sum n_i$

Remarque

Les durées d'éxécution des algorithmes qui vont suivre ne peuvent être comparées que au sein d'un même graphique puisques différents ordinateurs ont été utilisés pour des graphiques différents.

Les algorithmes de base:

L'algorithme glouton

L'implémentation L'algorithme glouton *src/strategy/greedy.rs* va chercher, parmi tous les mouvements possibles, celui qui permettra d'avoir le score instantané le plus élevé. Il suffit ainsi en rust de prendre le mouvement maximisant le score une fois un mouvement joué.

Evalutation des performances Il parait assez évident que ce genre de stratégie n'est pas optimale, mais elle a l'avantage d'avoir la rapidité la plus élevée. ($O(n_1)$)
On ne va donc pas comparer cet algorithme avec les autres puisque la solution renvoyée n'est optimale qu'avec une profondeur 1.

L'algorithme Min-Max

L'implémentation L'implémentation de l'algorithme Min-Max est faite d'après sa version pseudo-code disponible sur wikipedia.org. Cependant, nous avons cherché à implémenter une version plus performante en rust en utilisant les avantages que rust propose sur les itérateurs.
La version implémentée est la version non negamax (question de comprehension et de visualisation de l'algorithme : choix personnel)
Son implémentation se trouve dans *src/strategy/minmax.rs* #### Evalutation des performances

En reprenant les notations du cours, on a $W = n$ et $D = n$ puisque toutes les opérations se font séquentiellement dans l'algorithme de base. (dans le pire et meilleur cas)

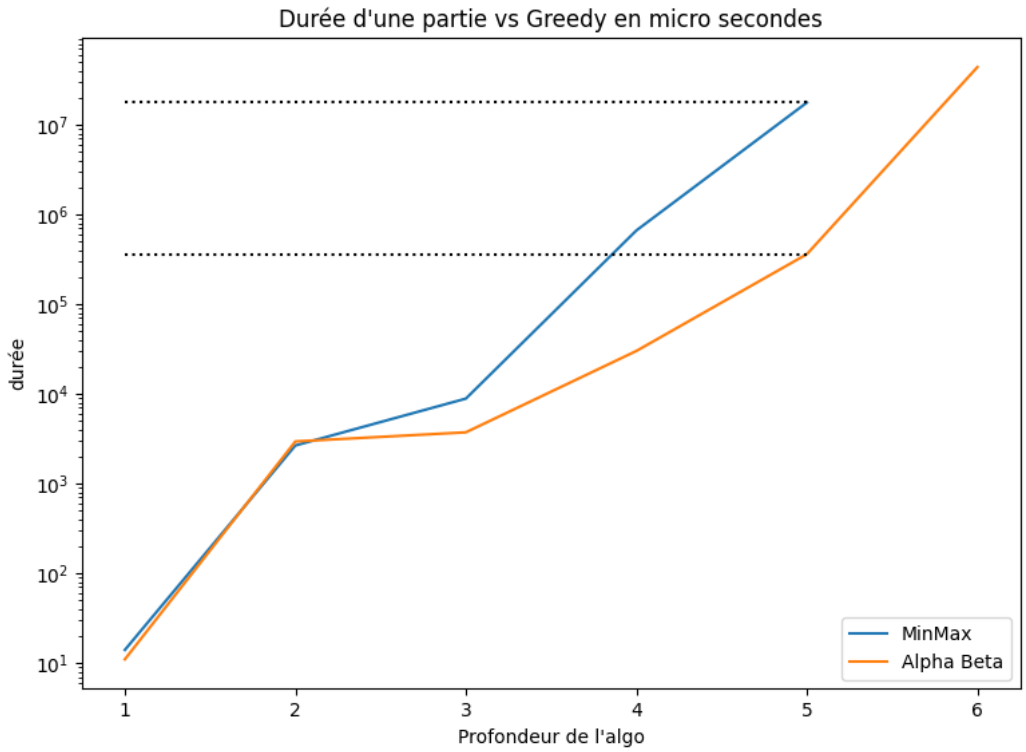
L'agorithme Alpha-Beta

L'implémentation L'implémentation de l'algorithme Alpha-Beta est également inspirée de la version pseudo-code disponible sur wikipédia. Encore une fois, nous avons implémenté la version non négamax.
Son implémentation se trouve dans *src/strategy/alphabeta.rs*

Evalutation des performances Cette fois-ci, le pire et meilleur cas seront différents. * Dans le pire cas, on se retrouve avec les valeurs de l'algorithme min-max, c'est-à-dire $W = n$ et $D = n$ * Dans le meilleur cas, il y a moins de nœuds à explorer. Comme le nombre de fils qu'a un nœuds n'est pas fixe, on ne peut pas donner de valeur exacte, mais on se retrouve avec $n' \leq n$ et $W = n'$ et $D = n'$

Comparaison Alpha-Beta vs MinMax

Nous avons effectué des mesures de performances pour comparer les algorithmes Alpha-Beta et Min-Max
Des parties ont été effectuées avec comme joueur 1 soit Alpha-Beta soit Min-Max et avec comme joueur 2 Greedy. Les deux algorithmes renvoient le même mouvement à chaque étape (vérifié expérimentalement).



On remaque sur la figure 1 que l'algorithme Alpha-Beta est effectivement plus efficace que l'algorithe Min-Max. On peut par exemple remarquer que Alpha-Beta est 50x plus rapide que min-max pour une profondeur de 5.

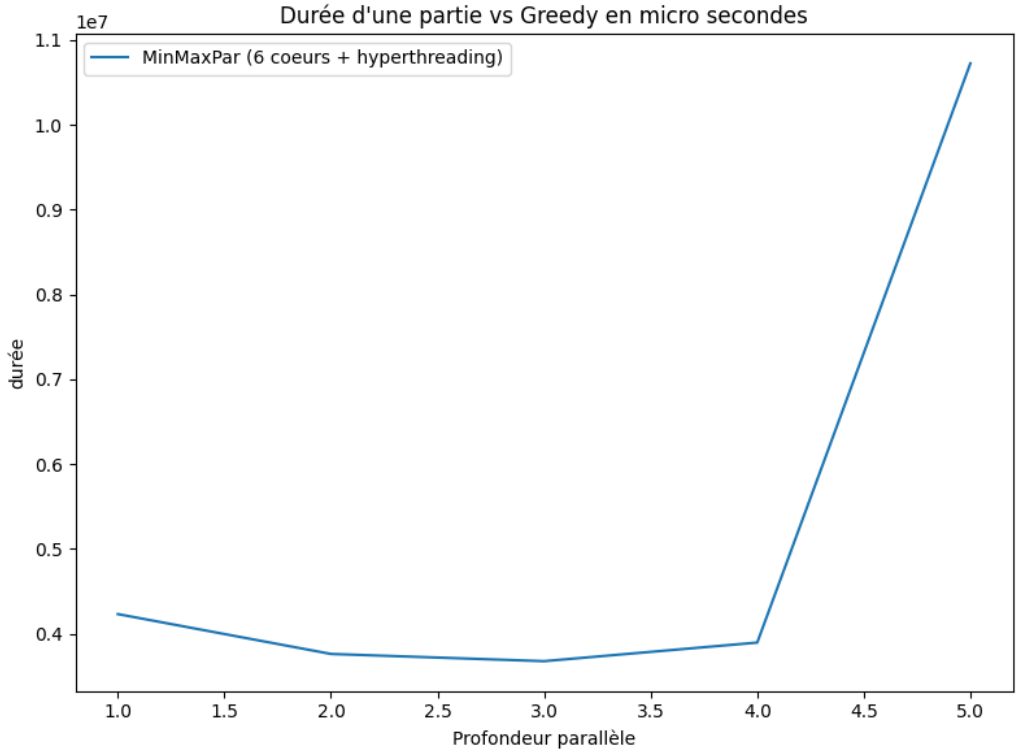
Les optimisations

Min-Max parallèle

L'implémentation Comme l'évaluation de la valeur de chaque fils d'un nœud est indépendante de la valeur des autres fils, on peut effectuer ces calculs en parallèle.
On peut alors créer une version parallèle de l'algorithme Min-Max.
Son implémentation se trouve dans *src/strategy/min_max_parallel.rs*
L'idée est de créer une nouvelle tache en parallèle pour chaque fils tant qu'on n'a pas atteint une profondeur limite donnée, et de finir avec un algorithme Min-Max classique au dela de cette profondeur.

Evalutation des performances On va noter p la profondeur max de l'algorithme parallèle.
On a alors $W = n$ et $D = p + D'$ avec $D' = \max_D(D_{\text{sous arbre de racine à la profondeur p+1}})$.
On peut alors remarquer rapidement que $D_{\text{parallèle}} < D_{\text{séquentiel}}$

Mesurons les performances en pratique:
* Trouvons expérimentalement pour quelle valeur de profondeur de parallélisme l'algorithme fonctionne le mieux (moyenne sur 3 mesures de perf)



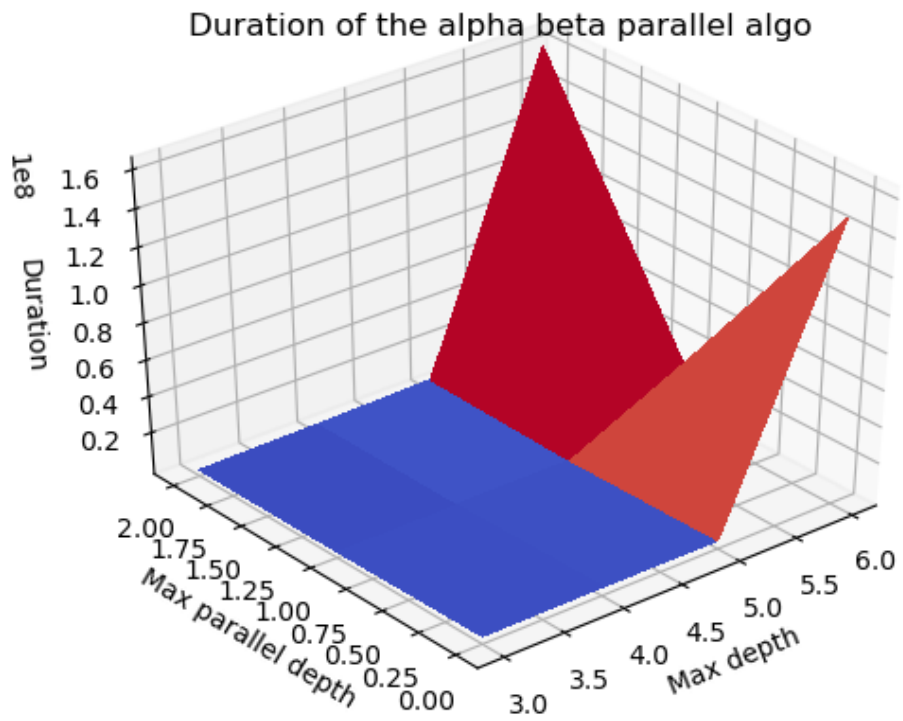
On voit sur la figure 2 qu'une limitation de parallélisme à la profondeur 2 ou 3 semble la plus efficace. On prendra par la suite la valeur 3.

Alpha-Beta parallèle : approches

L'algorithme Alpha-Beta est, par sa structure, séquentiel.
On peut avoir plusieurs approches pour essayer de faire une version parallèle:
* On peut imaginer utiliser des mutex pour assurer des écritures et lectures valides des valeurs α et β. On pourrait alors de manière similaire à l'algorithme Min-Max parallèle calculer la valeur de chaque état de façon parallèle jusqu'à une profondeur donnée et terminer de façon séquentielle (en utilisant toujours des mutex). Cependant, utiliser des mutex implique forcément des threads qui seraient bloquées et mis en attente, et donc les performances seront réduites.
* Une autre approche consisterait à calculer une valeur de α et de β sur quelques premiers fils de l'état initial de façon séquentielle et d'utiliser ces valeurs comme initialisation pour les autres fils, pour lesquels, cette fois-ci, on pourrait évaluer leur valeur de façon parallèle en considérant que chaque thread a ses propres α et β qui ont déjà une valeur assez grande pour espérer élaguer rapidement les sous-arbres. * Finalement, la version implémentée par manque de temps et simplement parcourir chaque fils jusqu'à une profondeur donnée et d'ensuite initialiser un Alpha-Beta séquentiel avec

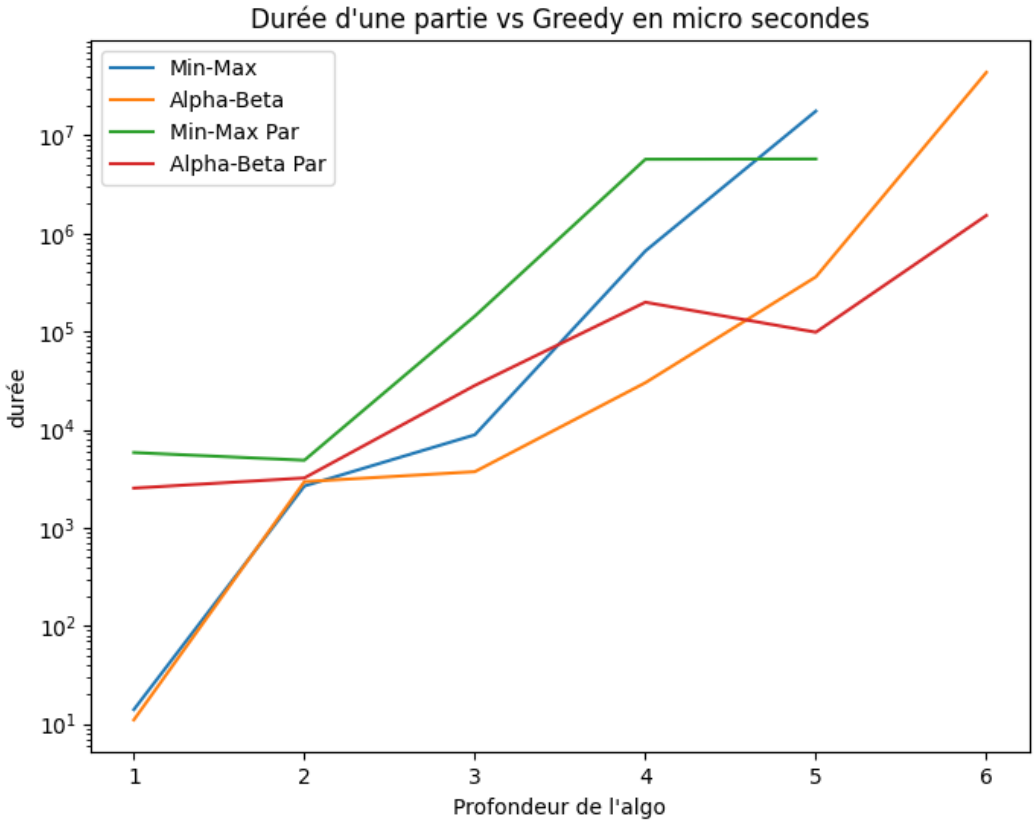
$\alpha = -\infty$ et $\beta = +\infty$. Cela permet de garder l'avantage d'Alpha-Beta (élagage) tout en faisant une partie en parallele.

Pour déterminer la profondeur parallèle, nous avons encore une fois fait des mesures de performance et nous avons trouvé une profondeur parallèle de 1 (figure 3)



Le comparatif final

On peut alors tester chaques algorithmes pour les comparer les uns aux autres. On va ici comparer l'algorithme Min-Max, Alpha-Beta et leur version parallèle contre greedy :



(les algorithmes n'ayant pas de valeur pour la profondeur 6 sont juste trop lents pour la mesure de performances)
De ces résultats, on peut pourrait définir un algorithme mixte qui utilise pour une profondeur maximale de 1 un des deux algorithmes séquentiels et sinon Alpha-Beta parallèle.