RAPPORT PUISSANCE 4 ET MCTS

Michel Wolkowicz
Rémy Mulhauser

I. Explication et implémentation du jeu et de l'algorithme

Nous avons implanté l'algorithme MCTS pour le jeu puissance 4.

Nous avons choisis de programmer en Java plutôt qu'en C car nous étions plus à l'aise en Java. (Vous trouverez ci-joint nos fichiers source)

Comment jouer ?

Pour exécuter le programme il faut simplement lancer le fichier Puissance4.jar.
Une fois ce programme lancé dans la fenêtre du terminal, il faut choisir qui de l'ordinateur ou du joueur va commencer; pour se faire taper 1 puis entrée pour que l'ordinateur commence, ou 0 puis entrée pour que le joueur commence.
Ensuite à chaque tour de l'humain, il faudra choisir dans quelle colonne, l'humain veut placer son pion puis entrée. On attend que l'ordinateur joue son tour et ainsi de suite.

Nous avons utilisé l'algorithme vu en cours:

- Sélection (treePolicy())
- Expansion (expand())
- Simulation (defaultPolicy())
- BackPropagation (backUp())

Pour modifier le temps de réflexion de l'ordinateur, il suffit de modifier le paramètre passé à la fonction ordiJoueMCTS, appelé dans le main (ligne 43).

exemple : e.ordiJoueMCTS(3) : 3 étant la limite de temps.

II. Questions

1) Nous avons affiché après chaque coups joués par l'ordinateur:

les coups possibles pour l'ordinateur
la récompense associée à chacuns de ces coups
le nombre de simulations pour chaques coups disponible
la probabilité de victoire pour chaques coups disponible
le nombre de simulations total

2) (Les tests sont réalisés sur un ordinateur avec un processeur à 3.5 GHz)

- Test 1 = limite de temps à 3 secondes: environ 630 simulation au premier coup de l'ordinateur (quand il commence). L'ordi joue très souvent le meilleur coup possible mais pas à tous les coups. On constate qu'au premier tour quand il commence , il placera son pion au milieu du plateau dans la colonne 2 ,3 ou 4 car c'est là où il a le plus de chance de gagner. Avec une limite de 3 seconde c'est impossible de gagner contre l'ordinateur.

- Test 2 = limite de temps à 1 seconde: environ 350 simulation au premier coup de l'ordinateur (quand il commence). On constate dès le début qu'il ne fait pas le meilleur choix , au premier tour quand il commence, l'ordi à plus de chance de gagner si il place le pion au milieu du plateau c'est à dire dans la colonne 2, 3, ou 4. On constate qu'il ne le place que rarement dans cette position au départ : parfois il le place en 1 ou 6. Cependant il peut encore gagner si le joueur ne l'en empêche "pas trop".

- Test 3 = limite de temps à 5 seconde: environ 820 simulation au premier coup de l'ordinateur (quand il commence). On constate que l'ordi joue tout le temps le bon coup et surtout pour nous empêcher de gagner : il nous bloque systématiquement quand nous avons 3 pions alignés. Nous n'avons pas gagnés une partie sur 6 de jouées contre l'ordi avec une limite de 5 secondes. (5 défaites 1 match nul).

- Test 4 = limite de temps à 4 seconde: environ 740 simulation au premier coup de l'ordinateur (quand il commence). Ce n'est pas possible de gagner contre l'ordinateur avec une limite de temps de 4 secondes.

3)
amélioration réalisé dans cette question :
Lors des simulations , au lieu de prendre un noeud fils au hasard et de l'explorer entièrement, on privilégie un noeud menant directement à un coup gagnant (quand cela est possible).
On constate que les probabilités de victoire sont plus élevés (dès le premier coup de l'ordinateur), ainsi que les récompenses associées à ces coups :

Sans cette amélioration lors du premier coup : récompenses entre 15 et 30 et des probabilités de victoire d'environ 0.5

Avec cette amélioration lors du premier coup : récompenses entre 27 et 34 et des probabilités de victoire d'environ 0.8

L'ordinateur paraît plus "redoutable" qu'avant , les coups qu'il joue empêche plus le joueur de construire une ligne de 4 (et même de 3), et on a l'impression qu'il joue de manière plus astucieuse.
Ce changement est lié au fait que l'on cherche à simuler des coups gagnants sans trop s'attarder sur des coups qui ne mèneront pas l'ordi à la victoire, en effet l'arbre obtenu à la fin de ces simulation comporte nécessairement un nombre de feuilles avec un état final ou l'ordinateur gagne qui est plus important que sans l'amélioration.

4)
Nous travaillons en Java , cependant , l'option gcc -03 est liée à l'optimisation:
il existe plusieurs niveau de cette optimisation :
-01 : optimisation basique, meilleur pour débugger.
-02 : une bonne optimisation , améliore les performances.
-03 : meilleurs performances , peut diminuer dans certains cas.

Avantage de l'option -03 :

Améliore considérablement la vitesse pour les petits projets

5)
Dans notre programme , on sélectionne le meilleur fils du noeud courant dans la fonction bestChild via la formule de l'algorithme UCT.
Afin de tester les critères robuste et max, on va changer cette fonction de sélection:
max : en choisissant le fils qui à la plus forte récompense
robuste : en choisissant le fils qui a le plus grand nombre de simulation
Max-Robuste (les deux à la fois) est bien plus performant que l'un ou l'autre seul.

```
// CRITERE DE SELECTION : ROBUSTE-MAX
t1 = recompense/nbSimulChild;
t2 = c * Math.sqrt(2*Math.log(nbSimulParent)/nbSimulChild);
t3 = t1 + t2;
// CRITERE DE SELECTION : MAX
t3 = child.getRecompense();


// CRITERE DE SELECTION : ROBUSTE
t3 = child.getNbSimulation();
```

6)
Complexité en temps de l'algorithme Minmax :

O(b^m)

avec b = nombre de coups possible à chaque état(point)
m = profondeur maximale de l'arbre
Cet algo réalise une exploration en profondeur d'abord complète de l'arbre de jeu(pas de limitation de profondeur)

Pour le premier coup notre graphe a (avec une limite de temps de 3 secondes) :

- environ 4500 noeuds
- une profondeur moyenne d'environ 15 (varie selon les simulations)
- un facteur de branchement de 7

Ce qui nous donne une estimation en temps de calcul de :
O(7^15) = 4 747 561 509 943
O(7^10) = 282 475 249 (Lorsqu'on à une profondeur de 10)