

UE Apprentissage par renforcement - séance 3

Valentin Emiya

M2 IAAA

11 décembre 2019

Cette semaine

AR et jeux : Monte Carlo Tree Search

- Un aperçu des techniques pour concevoir des joueurs automatiques (théorie des jeux, apprentissage) et des résultats
- MCTS : un cadre permettant de jouer à des jeux difficiles
- UCT : UCB + MCTS
- RAVE : encore plus efficace en pratique
- TP sur Othello

Remerciements

Ce cours reprend celui de Liva Ralaivola donné en 2018 en M2 IAAA.

Sommaire

- 1 Petit historique
- 2 Notion de bases
- 3 Monte Carlo tree search (MCTS)
 - Le cadre MCTS
 - UCT : UCB pour les arbres de jeu
 - Rapid Action Value Estimation (RAVE)
- 4 TP
- 5 Conclusion

Sommaire

- 1 Petit historique
- 2 Notion de bases
- 3 Monte Carlo tree search (MCTS)
 - Le cadre MCTS
 - UCT : UCB pour les arbres de jeu
 - Rapid Action Value Estimation (RAVE)
- 4 TP
- 5 Conclusion

Petit historique

Humain vs Machine (en 2013)



MORPION



PUISSEANCE 4

1995



DAMES*

2007



OTHELLO

1995, 1997



SCRABBLE



1996 : Première victoire ordinateur

2005 : Dernière victoire humaine



POKER



BRIDGE



GO

Résolus
Ordi >> Humain

Ordi > Humain

Humain > Ordi

Petit historique

Humain vs Machine (en 2018)



MORPION



PUISSEANCE 4



OTHELLO

1995, 1997



SCRABBLE



DAMES*

2007



POKER



BRIDGE



ECHECS

1996 : Première victoire ordinateur
2005 : Dernière victoire humaine

GO

Résolus
Ordi>Humain

Ordi>>Humain

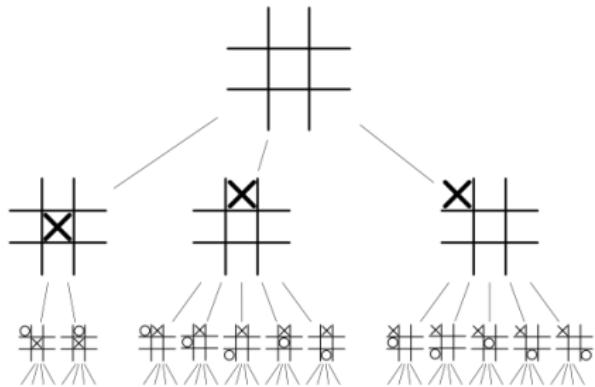
Ordi>Humain

Humain>Ordi

Sommaire

- 1 Petit historique
- 2 Notion de bases
- 3 Monte Carlo tree search (MCTS)
 - Le cadre MCTS
 - UCT : UCB pour les arbres de jeu
 - Rapid Action Value Estimation (RAVE)
- 4 TP
- 5 Conclusion

Arbre de jeu



(from Wikipedia)

Principe :

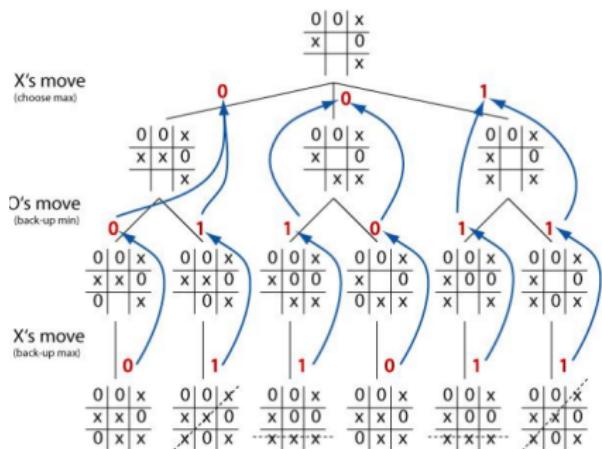
- chaque noeud : un état possible du jeu (=position, configuration)
- racine : état courant (e.g., début de la partie)
- fils : toutes les états accessibles via une action de jeu

Utilité : permet d'explorer toutes les parties possibles, d'analyser plusieurs coups à l'avance.

Problème : souvent trop grand, ne peut pas être construit entièrement (notion de facteur de branchement).

Minmax

From Elnagar et al., 2014



Principe :

- Alternance de tours "min" et "max" où les joueurs jouent optimalement
- Le joueur min sélectionne une action conduisant à la valeur min
- Le joueur max sélectionne une action conduisant à la valeur max

Utilité : permet de résoudre parfaitement un jeu si l'arbre est construit entièrement.

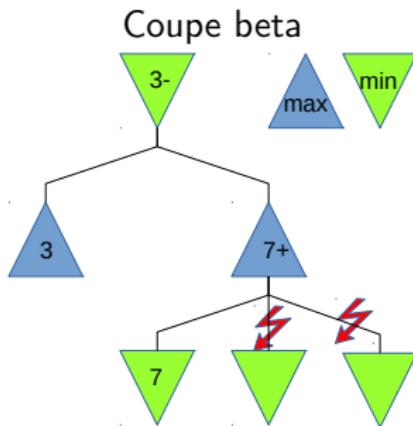
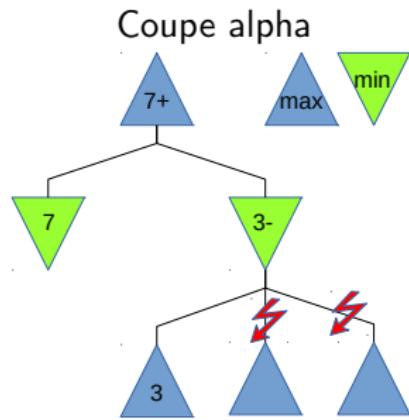
Problème : nécessite l'arbre entier.

Coupes Alpha-Beta : principe

Source :

http://lig-membres.imag.fr/pernet/Enseignements/L3METI_Prog00/MinMaxAlphaBeta.pdf

Idée : on peut résoudre le problème min-max sans explorer tout l'arbre.



Si la valeur d'un fils f d'un noeud min est inférieure à la valeur courante d'un noeud max ancêtre, alors les frères de f n'ont pas besoin d'être explorés.

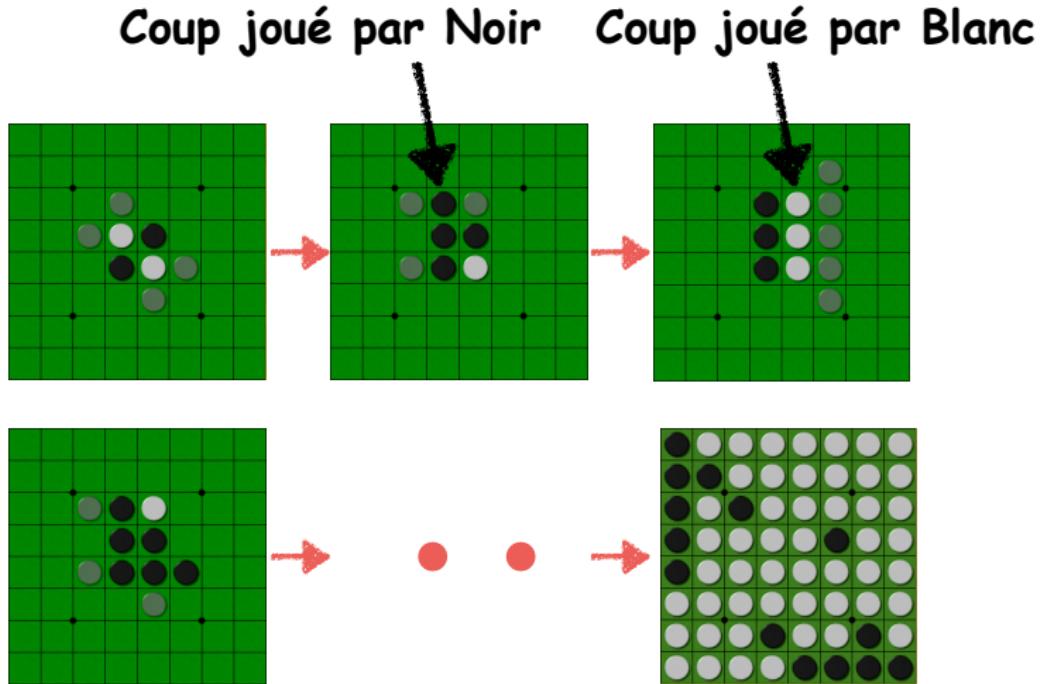
Si la valeur d'un fils f d'un noeud max est supérieure à la valeur courante d'un noeud min ancêtre, alors les frères de f n'ont pas besoin d'être explorés.

Coupes Alpha-Beta : demo

Voir :

http://inst.eecs.berkeley.edu/~cs61b/fa14/ta-materials/apps/ab_tree_practice/

Othello : les règles



Othello : quelques chiffres

- Plateau : 64 cases

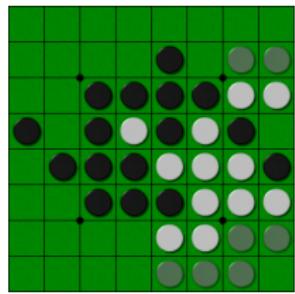
- Facteur de branchement : ≈ 8

$\approx 10.000.000.000.000.000.000.000.000 = 10^{28}$ configurations possibles
($\approx 10^{11}$ étoiles dans la Voie Lactée)
($\approx 10^{23}$ grains de sables sur Terre)

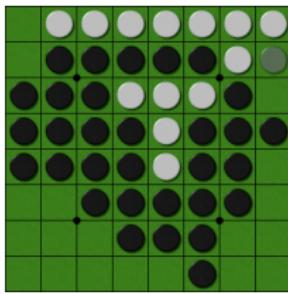
- Impossibilité de construire l'arbre de jeu entier

→ Nécessité d'une fonction d'évaluation

Notion de fonction d'évaluation



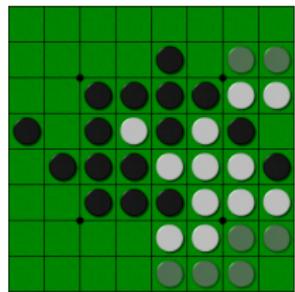
Victoire?
Défaite?



Victoire?
Défaite?

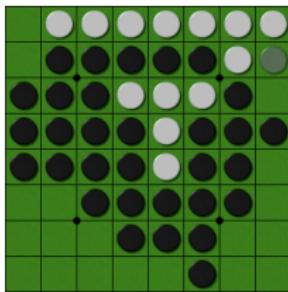
Estimer si un état est gagnant ou perdant ?

Notion de fonction d'évaluation



Victoire?

Défaite?



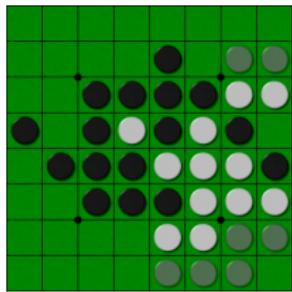
Victoire?

Défaite?

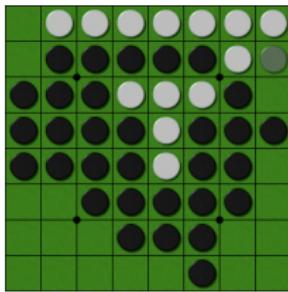
Estimer si un état est gagnant ou perdant ?

Estimer la qualité d'un état ?

Notion de fonction d'évaluation



Victoire?
Défaite?



Victoire?
Défaite?

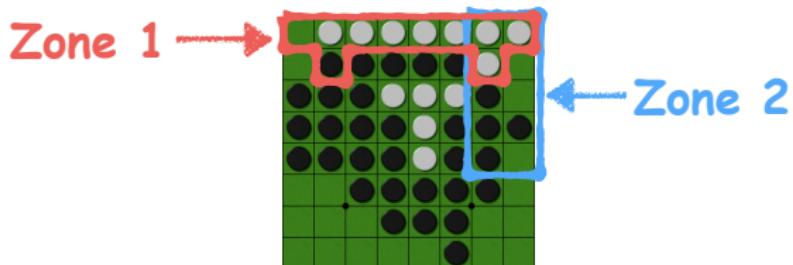
Estimer si un état est gagnant ou perdant ?

Estimer la qualité d'un état ?

Solutions :

- concevoir une fonction d'évaluation à partir de connaissances expertes
- apprendre une fonction d'évaluation

Logistello [Buro, 1998] : fonction d'évaluation d'un état

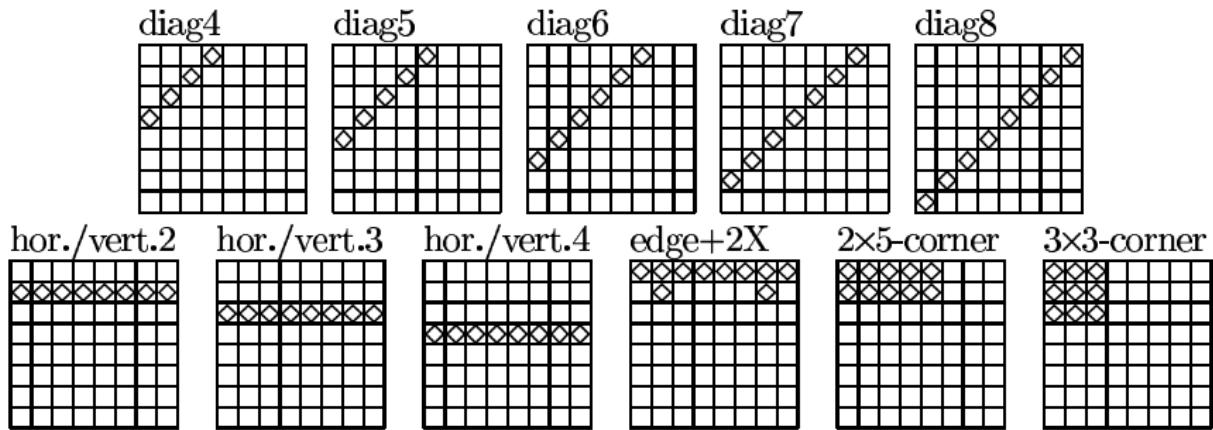


$$\text{Valeur}(\text{Board}) = 2 \times \text{force(zone 1)} + 10 \times \text{force(zone 2)} + \dots$$

Poids « appris » à partir
de millions de parties

The equation shows the evaluation function for a Go board. It multiplies the force of Zone 1 by 2 and the force of Zone 2 by 10, then adds the results. Below the equation, a green text explains that these weights were learned from millions of games.

Logistello [Buro, 1998] : fonction d'évaluation d'un état



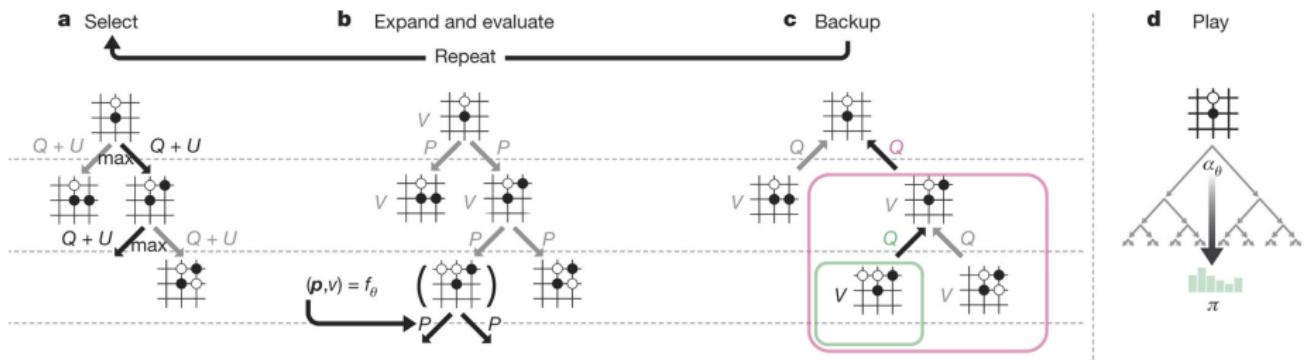
Valeur() = 2 x force(zone 1) + 10 x force(zone 2) + ...

Apprentissage

force(motif zone) ~ #app. motifs victoires / # motifs
 poids = régression logistique (1 minuscule réseau de neurones)

Renforcement et arbre de jeu : exemple d'AlphaGo

From *Mastering the game of Go without human knowledge*, D. Silver et al., Nature, 550, 2017.



MCTS in AlphaGo Zero. a, Each simulation traverses the tree by selecting the edge with maximum action value Q , plus an upper confidence bound U that depends on a stored prior probability P and visit count N for that edge (which is incremented once traversed). b, The leaf node is expanded and the associated position s is evaluated by the neural network $(P(s, .), V(s)) = f_\theta(s)$; the vector of P values are stored in the outgoing edges from s . c, Action value Q is updated to track the mean of all evaluations V in the subtree below that action. d, Once the search is complete, search probabilities π are returned, proportional to $N^{\frac{1}{\tau}}$, where N is the visit count of each move from the root state and τ is a parameter controlling temperature.

Sommaire

- 1 Petit historique
- 2 Notion de bases
- 3 Monte Carlo tree search (MCTS)
 - Le cadre MCTS
 - UCT : UCB pour les arbres de jeu
 - Rapid Action Value Estimation (RAVE)
- 4 TP
- 5 Conclusion

MCTS : aperçu

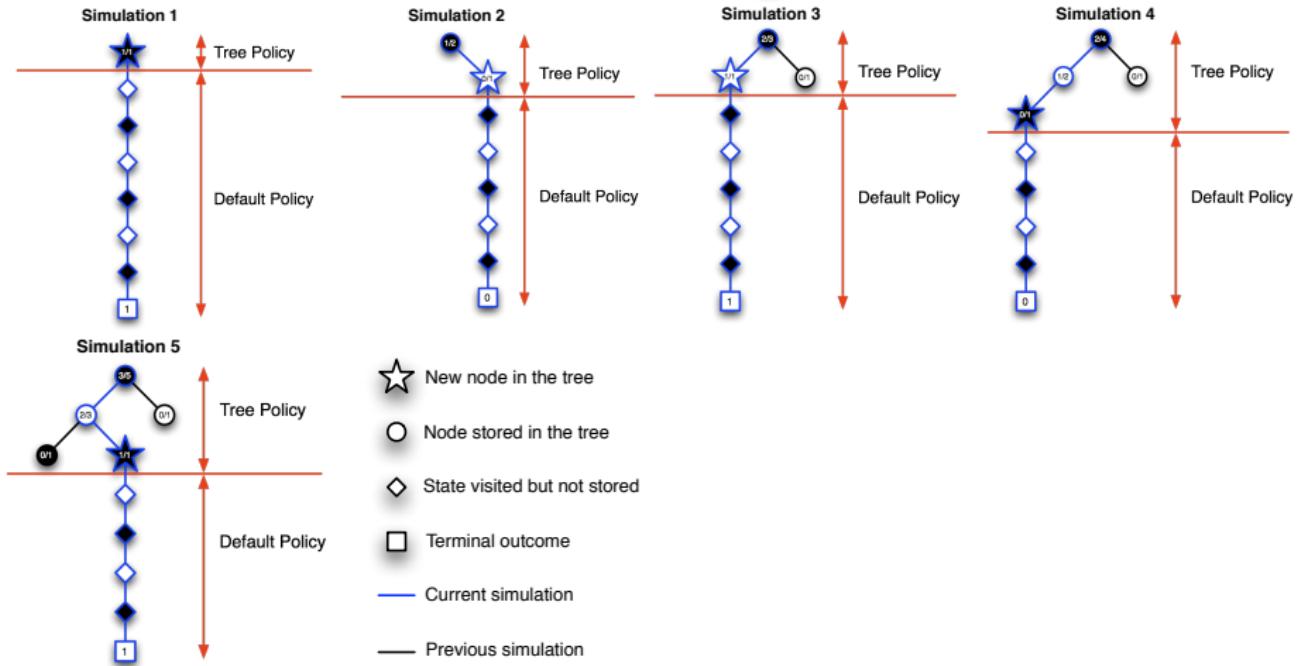
Contexte :

- arbre de jeu énorme.
- pas de bonne fonction d'évaluation disponible.
- parfaite connaissance de l'environnement : possibilité de simuler le jeu.

Idées :

- lancer de multiples simulations à moindre coût
- construire partiellement l'arbre en fonction des simulations selon un principe d'exploration/exploitation
- apprentissage par renforcement de la fonction d'évaluation
- MCTS de base + nombreuses possibilités d'amélioration

MCTS : exemple (from Gelly et al., JMLR, 2011)



MCTS : simulations en 4 étapes

Initialisation : arbre à un noeud \equiv état courant.

Répéter les simulations :

- ➊ Descente dans l'arbre actuel depuis la racine via une stratégie de descente (*tree policy*).

Exemple : ϵ -greedy

- ➋ Ajout d'un nouveau noeud

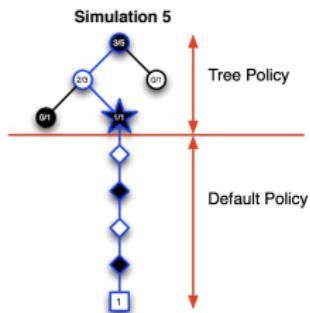
Dès qu'on rencontre un fils jamais exploré.

- ➌ Rollout : descente rapide via une stratégie par défaut (*default policy*)

Exemple : au hasard.

- ➍ Mise à jour de l'arbre actuel par rétropagation dans chaque noeud visité

Renvoyer : choisir l'action à jouer à la racine



Upper Confidence Tree (UCT) = MCTS + UCB

Principe : utiliser un bandit UCT dans chaque noeud de l'arbre.

Dans un noeud/état s et pour l'action a , on définit la **borne UCB**

$$q^+(s, a) = q(s, a) + c \sqrt{\frac{\log n(s)}{n(s, a)}}$$

avec

- $c > 0$ hyperparamètre,
- $n(s)$ nombre de visites dans s ,
- $n(s, a)$ nombre de fois que a a été sélectionné dans s ,
- $q(s, a) = \text{moyenne des gains obtenus dans } s \text{ en sélectionnant } a$.

Choix de l'action : $\pi(s) = \text{argmax}_a q(s, a)$.

Rapid Action Value Estimation (RAVE)

Principe : lorsqu'on observe une récompense obtenue dans un état s via une action a , on met à jour $q(s', a)$ et $n(s', a)$ pour tous les états s' où l'action a peut être choisie le long de la descente simulée depuis s .

Avantage : réduction de la variance sur les $q(s', a)$ car augmentation du nombre de valeurs moyennées.

Inconvénient : augmentation du biais du fait que $s' \neq s$

Sommaire

- 1 Petit historique
- 2 Notion de bases
- 3 Monte Carlo tree search (MCTS)
 - Le cadre MCTS
 - UCT : UCB pour les arbres de jeu
 - Rapid Action Value Estimation (RAVE)
- 4 TP
- 5 Conclusion

TP : grandes lignes

- Partir du code MTCS-UCT pour Othello
- Étudier l'évolution de la qualité du joueur en fonction du nombre de noeuds développés
- Améliorer le joueur en introduisant des connaissances du jeu (*UCT with prior knowledge*)
<https://hal.inria.fr/inria-00164003/document>
- Améliorer le joueur en utilisant RAVE
<http://www.cs.utexas.edu/~pstone/Courses/394Rspring13/resources/mcrave.pdf>

Sommaire

- 1 Petit historique
- 2 Notion de bases
- 3 Monte Carlo tree search (MCTS)
 - Le cadre MCTS
 - UCT : UCB pour les arbres de jeu
 - Rapid Action Value Estimation (RAVE)
- 4 TP
- 5 Conclusion

Conclusion

MCTS

- Contexte : espace d'états/actions très grand, simulations possibles
- Stratégie : simulations avec exploration/exploitation pour construction partielle de l'arbre de jeu
- Concepts utilisés : environnement avec états et actions, bandits/UCB, Monte Carlo

À venir : processus de décision de Markov (MDP)

Pour la semaine prochaine, se familiariser avec les bases des MDP en relisant le chapitre 13 du livre de Mitchell et en lisant la section 3.1 du livre de Sutton (voir liens sur Ametice).