IFT 608 / IFT 702
Planification en intelligence artificielle

Monte-Carlo Tree-Search

Professeur: Froduald Kabanza

Assistants: D'Jeff Nkashama & Jordan Félicien Masakuna

UDS Université de Sherbrooke

Sujets couvert

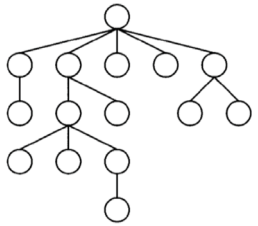
- **Approches de planification**
  (Barto & Sutton, 8.1) (Russel & Norvig, 3, 4, 5, 11)

- **Architecture de planification, exécution et apprentissage**
  (Barto & Sutton, 8.2) (Russel & Norvig, 2.4)

- ***Monte-Carlo Tree-Search***
  (Sutton & Barto, 8.11) (Russel & Norvig, 5.4)

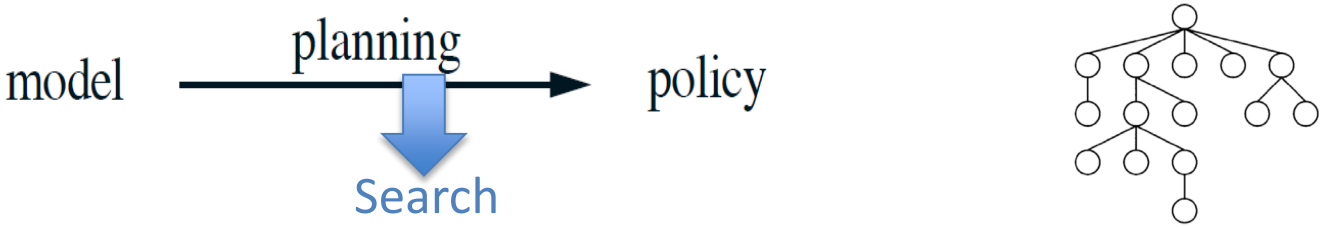
Planification

- Pour planifier, un agent a besoin d'un modèle de l'environnement
- Modèle = toute connaissance que l'agent peut utiliser pour prédire comment l'environnement va répondre à ses actions:
  - Modèle stochastique, *p(s', r |s,a),* pour la programmation dynamique avec *value-iteration*, *policy-iteration*
  - Modèle déterministe, pour A*
  - On verra d'autres représentations
- Par planification, on entend le processus (algorithme) qui prend un *modèle* comme entrée et produit une *politique* (plan, stratégie)

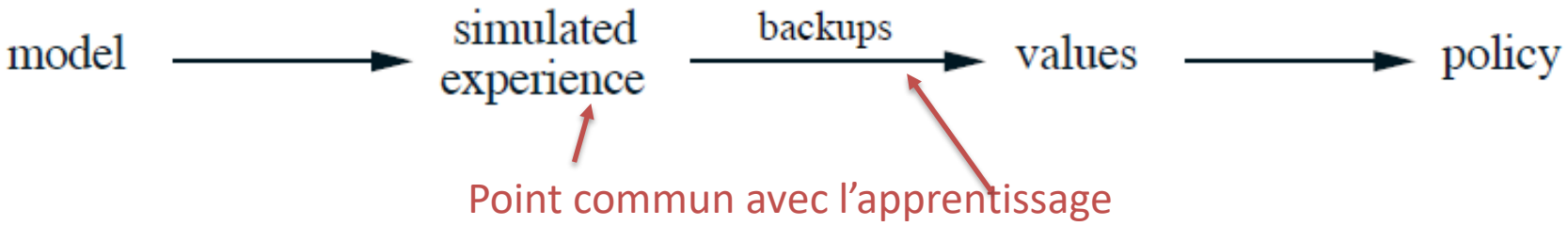


Planification

- Toutes les approches étudiées dans ce cours procèdent par une recherche heuristique dans un espace représenté par un arbre ou un graphe.

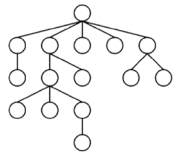


- Les approches state-space combinent des étapes de simulation de l'expérience et de calcul de la valeur de l'état (*backup*).



- Les approches plan-space transforme itérativement un plan initial jusqu'à obtenir un plan final.

Approches de planification



Les approches diffèrent selon la représentation de l'espace de recherche, du modèle, la représentation de la politique (plan, stratégie), le nombre d'agents, la technique utilisées pour explorer l'espace (recherche et optimisation) et le type de problèmes auxquelles elle s'appliquent (*path planning* vs *task planning*)

Space representation **State-Space** *vs* **Plan-Space**

Black-box

Vector, proposition, factored

Distribution (state vs set of states)

Rapidl-exploring Random Tree

Model Actions

Goal

Deterministic: (s,a,s') *vs* **Distribution:** P(s |a,s')

Reachable states *vs* **State sequence** *vs* **Reward function**

Plan representation **Tabular** *vs* **Parametarized** (e.g., neural network)

Agents **One** vs **Many** **Competing** vs **Cooperative**

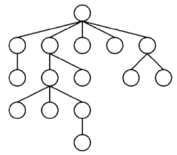
Search process **Simulated experience** **Iterative plan-transformation**

Backup value propagation Dynamic programming Trajectory sampling

Forward-chaining *vs* Backward-chaining Learning

Constraint solving Causal Hierarchical Heuristics

Approches de planification



Les approches diffèrent selon la représentation de l'espace de recherche, du modèle, la représentation de la politique (plan, stratégie), le nombre d'agents, et la technique utilisées pour explorer l'espace (recherche et optimisation).

Space representation **State-Space**

Vector, proposition, factored

Planning with temporal goals

Model Actions **Deterministic:** (s,a,s')

Goal **State sequence**

Plan representation **Tabular**

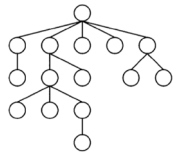
Agents **One**

Search process **Simulated experience**

Forward-chaining

Causal Heuristics

Approches de planification



Les approches diffèrent selon la représentation de l'espace de recherche, du modèle, la représentation de la politique (plan, stratégie), le nombre d'agents, et la technique utilisées pour explorer l'espace (recherche et optimisation).

Space representation

Plan-Space

Partially-Order HTN Planning

Vector, proposition, factored

Model

Actions

Goal

Deterministic: (s,a,s')

Reachable states

Plan representation

Tabular

Agents

One

Search process

Simulated experience

Iterative plan-transformation

Forward-chaining

Constraint solving

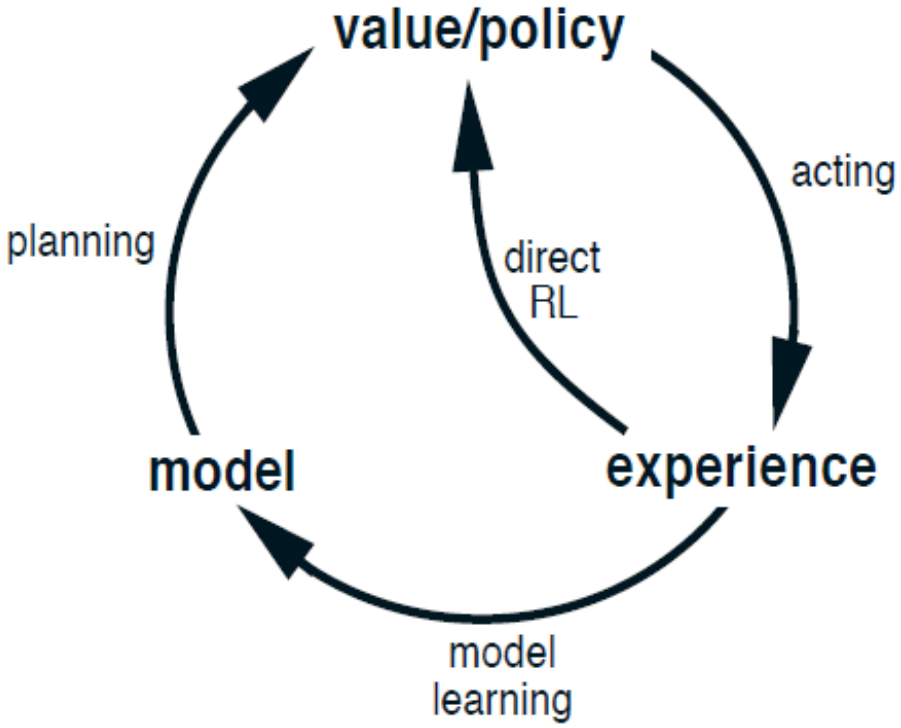
Causal

Hierarchical

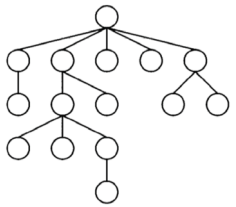
Heuristics

Planification et apprentissage intégrée

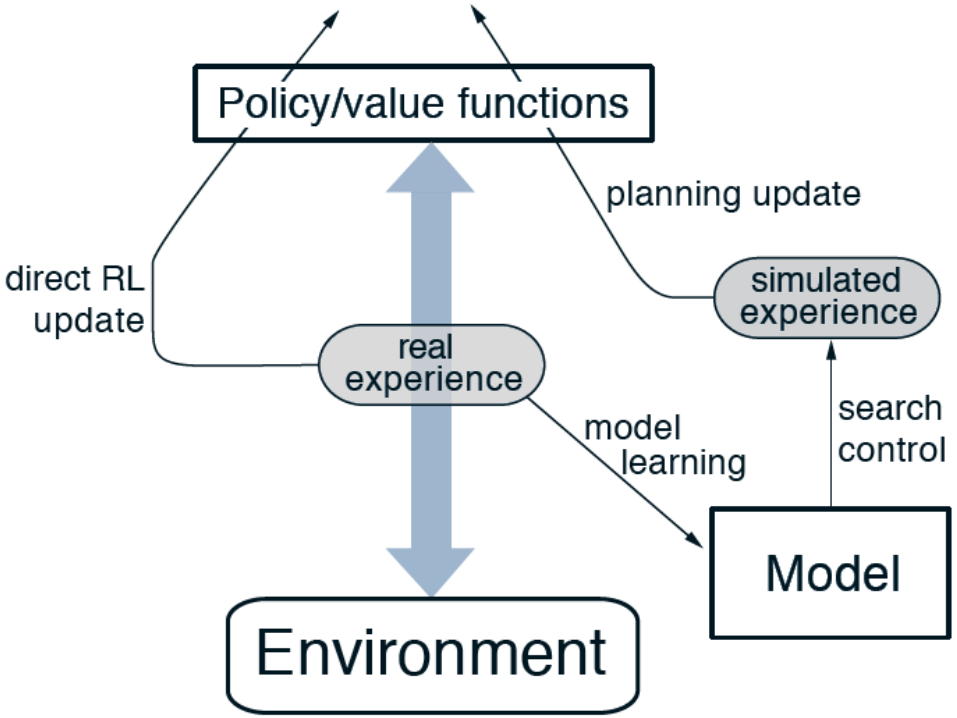
Architecture Dyna
(Sutton & Barto, 8.3)



Planification et apprentissage intégrée

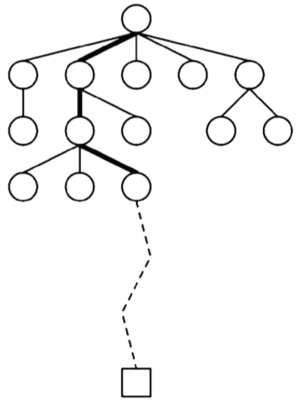


Architecture Dyna
(Sutton & Barto, 8.3)



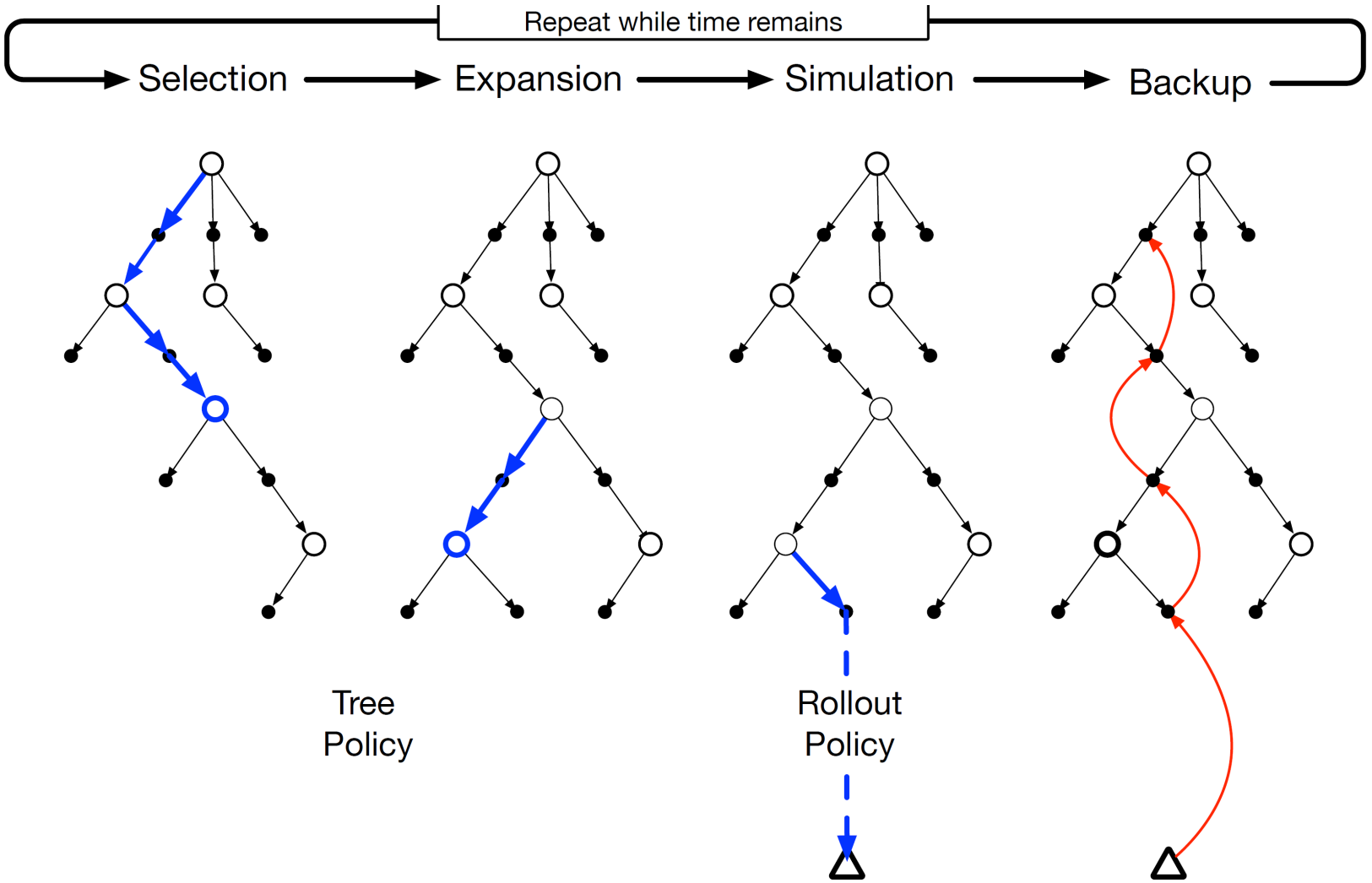
Principe de base de MCTS

- Le principe de Monte-Carlo Tree-Search (MCTS) est:
  - Parcourir (générer) l'espace d'états par **échantillonnage aléatoire.**
  - **Simuler** une partie complète pour évaluer la fonction d'utilité.



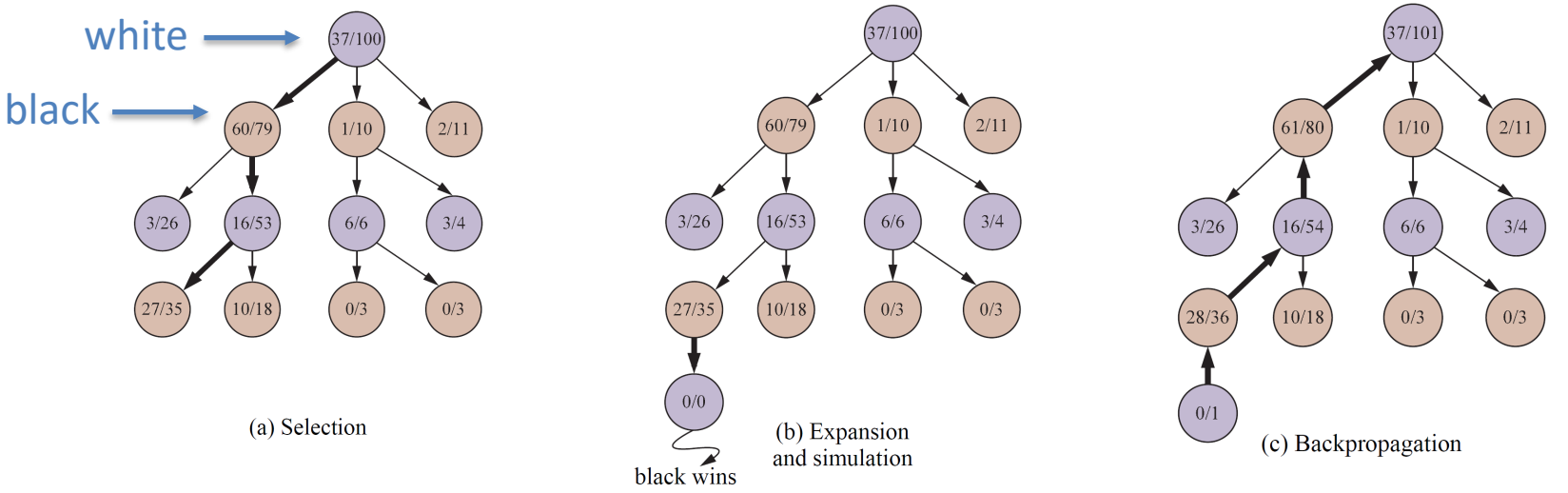
- À chaque itération:
  - **Sélection**: Commençant par la racine de l'arbre, en suivant une politique (*tree policy*) qui balance exploration vs exploitation, descendre à un nœud non encore complétement expansé (il existe un action non encore choisie)
  - **Expansion**: Agrandir l'arbre au nœud sélectionné, en choisissant une action non encore choisie en construisant un successeur correspondant.
  - **Simulation**: Simuler une partie complète à partir de l'enfant.
  - **Rétropropagation**: Mettre à jour l'utilité des nœuds

MCTS pour MDP (Sutton & Barto, 8.11)



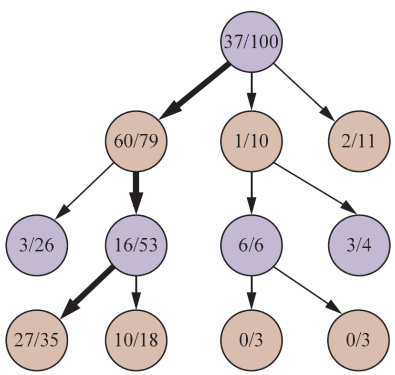
MCTS pour approximer *minmax* (Russel & Norvig, 5.4)

- Chaque est étiquetté par son **utilité**: *nombre de simulations gagnées / nombre total de simulations*
- **Sélection**: Commençant par la racine de l'arbre, en suivant une politique (*tree policy*) qui balance exploration vs exploitation, descendre à un nœud non encore complétement expansé (il existe un action non encore choisie)
- **Expansion**: Agrandir l'arbre au nœud sélectionnée, en choisissant une action non encore choisie en construisant un successeur correspondant.
- **Simulation**: Simuler une partie complète à partir de l'enfant.
- **Rétropropagation**: Mettre à jour l'utilité des nœuds
- **À la fin, choisit l'action avec le plus grand nombre de simulations**

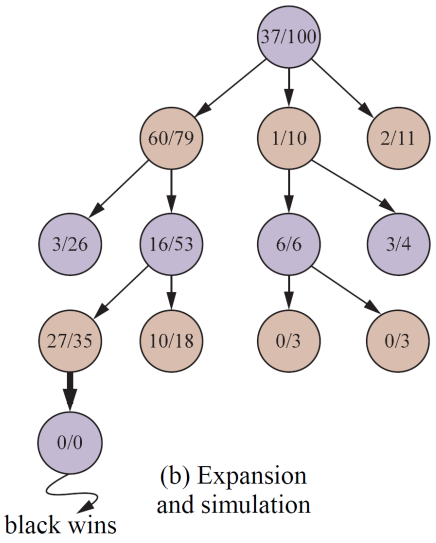


Algorithme MCTS (Russel & Norvig, 5.4)

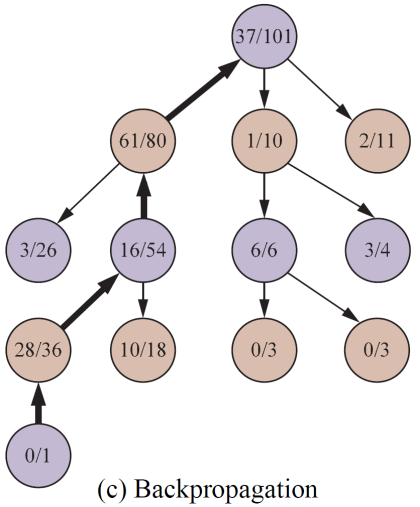
function MONTE-CARLO-TREE-SEARCH(*state*) **returns** *an action*
  tree ← NODE(*state*)
  while IS-TIME-REMAINING() **do**
    leaf ← SELECT(*tree*)
    child ← EXPAND(*leaf*)
    result ← SIMULATE(*child*)
    BACK-PROPAGATE(*result*, *child*)
  return the move in ACTIONS(*state*) whose node has highest number of playouts



(a) Selection



(b) Expansion and simulation



(c) Backpropagation

Algorithme UCT

- UCT (*Upper Confidence Bound on Trees*) est une version de MCTS qui utilise l'algorithme UCB1 (*Upper Confidence Bou*nd 1) pour implémenter la politique de sélection des nœuds permettant de résoudre le dilemme exploration vs exploitation

- *UCB1(s)=* $\frac{U(s)}{N(s)} + c * \sqrt{\frac{log\ N(Parent(s))}{N(s)}}$

  - *U(s)*: Utilité de l'état *s*
  - *N(s)*: nombre de simulations ayant passé par l'état s
  - *Parent(s)*: le parent de s
  - C : un hyper paramètre qui balance entre l'exploitation vs l'exploration. Théoriquement $\sqrt{2}$ mais en pratique choisi empiriquement.

- On pourrait utiliser UCB1 sur le Q-value en maintenant, Q(s,a) et N(s,a) au lieu de U(s) et N(s)

AlphaGo Zero & Alpha Zero (Sutton & Barto, 16.6.2)

AphaGo: Silver d. et al. Mastering the game of Go with DNNwand tree search. *Nature*, 2017
AlphaGo Zero: Silver d. et al. Mastering the game of Go without human knowledge. *Nature*, 2017
AlphaZero: Silver D. et al. Mastering Chess and Shogi by Self-Play with a General RL Algorithm. arXiv 1712.01815

- AlphaGo Zero combine l'*apprentissage automatique* avec MCTS
- Un **réseau de neurone f_θ** prend en entrée l'état *s* du jeu. Il a deux sorties (*p, v*):
  - **politique**: *p(a|s)* est la probabilité de choisir l'action *a* (inclus ne rien faire) dans l'état *s*.
  - **valeur**: *v(s)* est la probabilité (pour les noirs) de gagner à partir de l'état s.
- Le réseau est entrainé en jouant contre lui même pour générer des exemples (s_t, π_t, z_t): π_t est la politique suivie à partir de s_t et $z_t \in \{-1,1\}$ est le résultat de la partie jouée à partir de s_t (+1 si on gagne, -1 sinon).

  $(p,v) = f_\theta(s)$ avec la $loss(p, v, \pi, z) = (z - v)^2 - \pi \log p + c \|\Theta\|^2$

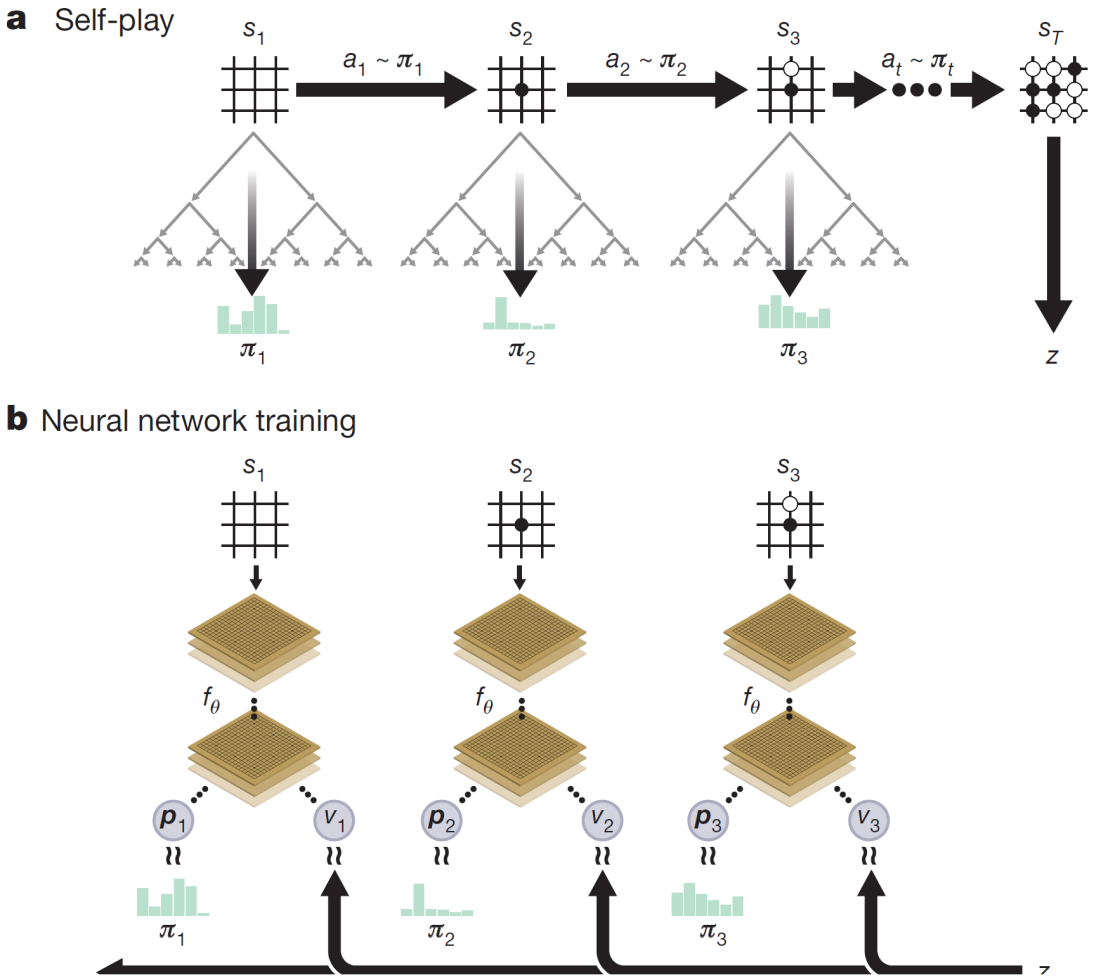
- La politique π_t est générée par UCT avec une politique de sélection

$$UCB(s,a) = Q(s,a) + c * \frac{p(a|s)}{1+N(s,a)}$$

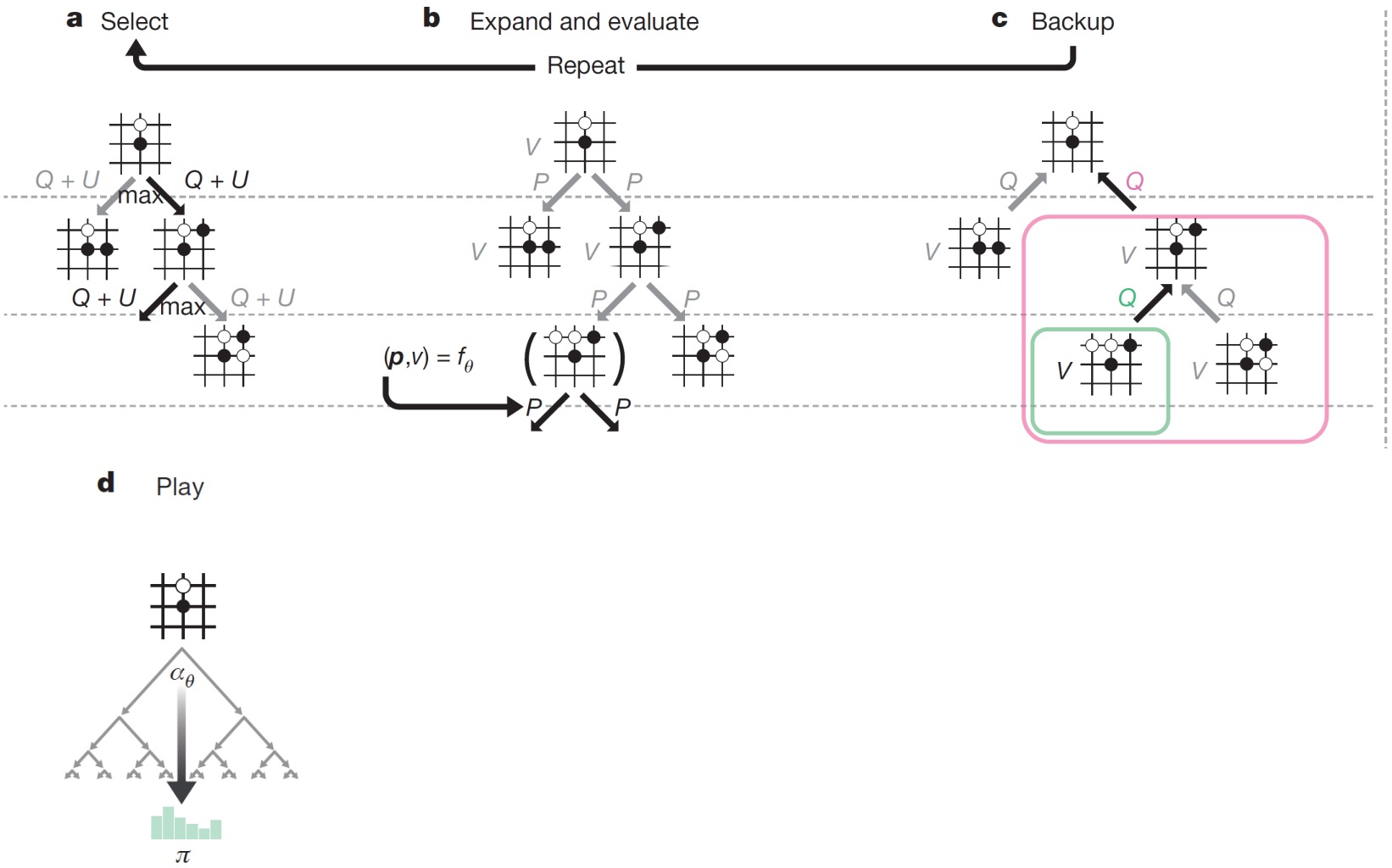
Noté *U(s,a)* dans le papier

- La politique de simulation est $p(a|s)$ définie par le reseau de neurones f_θ

AlphaGo Zero



AlphaGo Zero MCTS



Vous devriez être capable de...

- Expliquer l'algorithme *Monte-Carlo Tree-Search*