Monte Carlo Tree Search (MCTS)

Notes de cours

Nadjib Lazaar
Université Paris-Saclay
lazaar@lisn.fr

Introduction

Monte Carlo Tree Search (MCTS), Recherche arborescente Monte-Carlo, est un algorithme d'optimisation combinatoire pour la prise de décision dans les jeux et d'autres domaines. Il repose sur des simulations aléatoires pour explorer un espace de décisions et identifier les actions optimales. Cet algorithme est largement utilisé dans les jeux comme Go, Chess, et les jeux vidéo.

1 Histoire et évolution de MCTS

MCTS est une méthode d'exploration d'arbres qui combine des simulations statistiques basées sur les techniques Monte Carlo et des approches de recherche arborescente. Elle a été initialement développée pour résoudre des problèmes complexes, notamment dans les jeux à information parfaite tels que le Go.

Les méthodes Monte Carlo : Le concept de "Monte Carlo" est issu des méthodes probabilistes développées au cours des années 1940 dans le cadre du *projet Manhattan*. Ces techniques étaient conçues pour estimer des solutions approximatives à des problèmes où une recherche exhaustive était impossible.

1.1 Développement de MCTS dans les années 2000

Premières idées : L'idée de combiner des simulations Monte Carlo avec des arbres de recherche a émergé dans les années 1990, motivée par la nécessité d'améliorer les performances dans les jeux stratégiques.

Avancées clés dans les années 2006–2008 :

— **2006 :** MCTS a été formalisé dans des travaux sur les jeux combinatoires par Coulom, qui a introduit l'idée d'utiliser des simulations Monte Carlo pour guider la recherche.
— **2008 :** L'algorithme *Upper Confidence Bounds for Trees (UCT)*, proposé par Kocsis et Szepesvári, a marqué une étape majeure en intégrant une politique d'exploration-exploitation inspirée des bandits multi-bras.

1.2 Applications majeures

Le jeu de Go : L'une des percées les plus significatives pour MCTS a été son application au jeu de Go, où les algorithmes classiques comme Minimax échouaient en raison de la grande complexité combinatoire. Les programmes basés sur MCTS, comme `CrazyStone` et `MoGo`, ont atteint des niveaux de jeu compétitifs.

L'ère de l'intelligence artificielle : MCTS a également été utilisé comme composant clé dans des systèmes tels qu'`AlphaGo` (2016), qui combinait MCTS avec des réseaux neuronaux pour battre les meilleurs joueurs humains de Go.

1.3 Évolution et recherches actuelles

Aujourd'hui, MCTS est un sujet de recherche actif, avec des applications qui dépassent les jeux traditionnels pour inclure des domaines tels que :
— La planification en intelligence artificielle.
— Les systèmes de décision autonomes.
— L'optimisation des ressources dans les réseaux informatiques.
MCTS continue de jouer un rôle central dans le développement d'algorithmes d'IA modernes.

2 Concepts Clés de MCTS

MCTS repose sur quatre étapes principales, répétées pour explorer l'arbre de jeu :

1. **Sélection :** Parcours de l'arbre existant en sélectionnant les nœuds avec le plus grand potentiel selon un critère (comme UCB1).
2. **Expansion :** Ajout de nouveaux nœuds pour explorer des actions possibles non encore simulées.
3. **Simulation :** Exécution d'une partie simulée depuis un nœud jusqu'à un état terminal, souvent avec des actions choisies aléatoirement.
4. **Rétropropagation :** Mise à jour des statistiques des nœuds (gains et visites) sur le chemin de retour vers la racine.

3 Critère de Sélection : UCB1

Dans MCTS, le choix des actions à explorer repose sur un compromis entre **exploration** (examiner des actions moins explorées pour découvrir de meilleures options potentielles) et **exploitation** (concentrer les efforts sur les actions déjà prometteuses). Ce compromis est géré par le critère *Upper Confidence Bound 1* (**UCB1**).

3.1 Formule de UCB1

La formule utilisée pour le calcul de UCB1 est la suivante :

$$UCB1(a_i) = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N}{n_i}}$$

où :

- w_i : Le gain cumulatif obtenu en suivant l'action a_i.
- n_i : Le nombre de fois que l'action a_i a été simulée.
- N : Le nombre total de simulations effectuées depuis la racine.
- c : Un paramètre de contrôle, souvent réglé empiriquement, qui ajuste l'importance de l'exploration par rapport à l'exploitation.

3.2 Interprétation des termes

- **Premier terme : Exploitation**
  Le premier terme $\frac{w_i}{n_i}$ correspond à la moyenne des gains obtenus pour l'action i. Il favorise les actions qui ont montré de bonnes performances jusqu'à présent.
- **Second terme : Exploration**
  Le second terme $c\sqrt{\frac{\ln N}{n_i}}$ favorise les actions moins explorées. Il augmente lorsque n_i est faible, encourageant ainsi à explorer ces actions.
- **Paramètre c**
  Le paramètre c joue un rôle crucial dans l'équilibre entre exploration et exploitation :
  - Si c est grand, le poids de l'exploration augmente, ce qui est utile dans des environnements très incertains.
  - Si c est petit, l'algorithme privilégie l'exploitation des actions qui semblent déjà prometteuses.

3.3 Avantages de UCB1

Le critère UCB1 présente plusieurs avantages clés :

- Il est simple à implémenter et efficace en pratique pour équilibrer exploration et exploitation.
- Il garantit asymptotiquement que les actions les plus prometteuses seront explorées suffisamment souvent.

— Il s'adapte dynamiquement au nombre total de simulations N, ce qui en fait une méthode robuste pour une large gamme de problèmes.

3.4 Applications de UCB1 dans MCTS

Dans MCTS, UCB1 est utilisé à chaque étape de sélection de l'arbre. L'algorithme explore les actions disponibles à partir du nœud courant en choisissant celle qui maximise la valeur de UCB1(a_i). Ce mécanisme permet à l'algorithme de construire un arbre de recherche orienté vers les zones les plus prometteuses de l'espace des solutions.

3.5 Limites et extensions

Bien que UCB1 soit efficace, il présente certaines limites :

— Dans des environnements avec des distributions de récompenses complexes, il peut être nécessaire d'ajuster finement le paramètre c.

— Il suppose que les récompenses sont normalisées, ce qui peut ne pas être le cas dans toutes les applications.

Pour surmonter ces limites, plusieurs extensions ont été proposées, telles que UCB-Tuned et des critères adaptés à des environnements spécifiques.

4 Algorithme MCTS

Algorithme 1 : MCTS Algorithm

Input : s_0 : Initial state ; n : Number of iterations.
Output : `bestAction` : The best action to take from the root state.

begin
  for $i \leftarrow 1$ ***to*** n **do**
    $s \leftarrow$ `Selection`(s_0)
    $s' \leftarrow$ `Expansion`(s)
    `result` $\leftarrow$ `Simulation`(s')
    `Backpropagation`$(s',$ `result`$)$
  return `BestAction`(s_0)

Function `Selection`(s) **:**
  while $\forall s' \in child(s) : isVisited(s')$ **do**
    $s \leftarrow BestUCB\text{-}Child(s)$
  return s `// Return the first unexplored node or the most promising leaf node`

Function `Expansion`(s) **:**
  pick $s' \in child(s) : \neg isVisisted(s')$
  if $s' \neq nil$ **then** $setAsVisited(s')$
  return s' `// Return the newly created child node`

Function `Simulation`(s) **:**
  while $\neg isLeaf(s)$ **do**
    pick $s' \in child(s)$
    $s \leftarrow s'$
  return EVAL(s)

Function `Backpropagation`$(s,$ `score`$)$ **:**
  while $s \neq nil$ **do**
    $update\_nbVisits(s)$
    $update\_reward(s,$ `score`$)$
    $s \leftarrow parent(s)$
  return `// Backpropagation is complete`

Function `BestAction(root)` **:**
  $s^* \leftarrow \text{argmax}_{s' \in child(s)}(nbVisits(s'))$
  $a^* \leftarrow a_i : T(s_0, a_i) = s^*$
  return a^* `// Return the action corresponding to the most visited child of root`