Fiche résumé

lundi 19 septembre 2022 20:37

Algorithmes de recherche "aveugles"

- **Largeur d'abord** : garantie de trouver une solution (avec le moins d'actions possibles) mais demande beaucoup de mémoire
- **Coût uniforme** : idem largeur d'abord, mais trouve la solution de coût minimal (mais peut-être plus d'actions) en creusant les nœuds qui minimisent le coût $g(noeud)$ entre la racine et le nœud
- **Profondeur d'abord** : plus efficace en terme de mémoire, peut trouver une solution rapidement mais peut aussi perdre beaucoup de temps dans des branches inintéressantes
- Mélange : augmenter progressivement la profondeur maximale d'une recherche en profondeur d'abord

Algorithmes de recherche "informés" : le meilleur d'abord

Avec une heuristique $h(noeud)$ qui estime le coût restant pour le meilleur chemin entre *noeud* et *but*

- **Recherche gloutonne** : creuser le nœud qui minimise $h(noeud)$ pour se rapprocher le plus possible du but ; mais sans garantie sur le coût de la solution
- A^* : combiner coût uniforme et recherche gloutonne : creuser le nœud qui minimise $f(noeud) = g(noeud) + h(noeud)$ = estimation du coût total d'une solution passant par *noeud*

A^* : heuristiques admissibles

- Une heuristique est dite **admissible** si $h(noeud) \leq$ coût du meilleur chemin de *noeud* à *but* pour tout *noeud*
- Si l'heuristique est admissible alors A^* est garanti de trouver une **solution optimale**

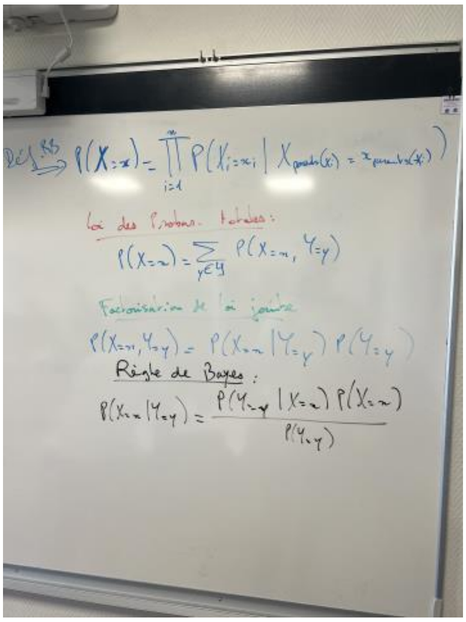
Algorithme générique

Squelette identique pour tous les algorithmes de recherche :

1. Prendre un nœud dans la file d'attente
2. Vérifier si le but est atteint à ce nœud (si oui, retourner la solution)
3. Développer le nœud et ajouter les fils à la file d'attente

Un algorithme de recherche = algorithme générique + fonction pour ordonner la file d'attente

- Largeur d'abord : ajouter les nouveaux nœuds à la fin de la file
- Coût uniforme : trier la file par ordre croissant de $g(noeud)$
- Profondeur d'abord : ajouter les nouveaux nœuds au début de la file
- Glouton : trier la file par ordre croissant de $h(noeud)$
- A^* : trier la file par ordre croissant de $f(noeud) = g(noeud) + h(noeud)$



Algo de jeux à deux joueurs :

- Sans hasard

MiniMax

AlphaBeta : reduire facteur de branchement moyen

Elaguer l'arbre pour gagner du temps

- L'arbre est parcouru en profondeur d'abord ⇒ la valeur des nœuds d'une branche peut être calculée avant de passer à la branche suivante
- Ne pas explorer les branches qui ne seront jamais jouées car elles ne correspondent pas à des stratégies optimales pour les joueurs

Algorithme Alpha-beta : minimax modifié (init : $\alpha = -\infty, \beta = \infty$)

Fonction **valeurMax** (nœud, α, β)

- $v = -\infty$
- Pour chaque *fils* de nœud
  - calculer sa valeur et le max courant $v = \max(v, \textbf{valeurMin}(fils, \alpha, \beta))$
  - Si $v \geq \beta$, retourner v
  - $\alpha = \max(\alpha, v)$
- Retourner v

Fonction **valeurMin** (nœud, α, β)

- $v = +\infty$
- Pour chaque *fils* de nœud
  - calculer sa valeur et le min courant $v = \min(v, \textbf{valeurMax}(fils, \alpha, \beta))$
  - Si $v \leq \alpha$, retourner v
  - $\beta = \min(\beta, v)$
- Retourner v

Limitation de profondeur : réduire prof d'une branche /!\ l'algo n'est maintenant plus optimal !

La fonction d'évaluation veut limiter la profondeur de l'arbre :

Au morpion, on peut choisir quelque chose comme

E = eval(X) - eval(O)

eval : w1*Phi1 + w2*Phi2

Phi1 = nb de lignes avec 1*X et 0*O

Phi2 = Nb de lignes avec 2*X et 0*O

W1 = 1

W2 = 3

Fonction d'évaluation linéaire

$$evaluation(etat) = \sum_{k=1}^{p} w_k \phi_k(etat)$$

Ex. aux échecs : $\phi_k(etat)$ = nb de pièces du type k, w_k = valeurs des pièces

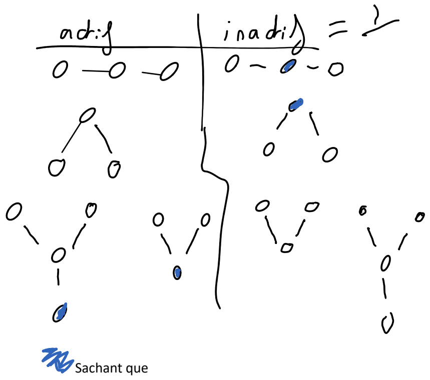
- Avec hasard

Expectimax

- Algorithme Minimax modifié tel que la valeur remontée soit
  - la valeur minimale sur l'ensemble des fils si c'est à Mini de jouer
  - la valeur maximale sur l'ensemble des fils si c'est à Max de jouer
  - la **valeur moyenne** sur l'ensemble des fils si c'est un **nœud chance** :

Monte Carlo : algo qui test plus souvent une solution qui parait très bonne

Rappel cours :



Sachant que