Intelligence Artificielle

Les jeux

Laurent.Bougrain@loria.fr

Introduction

Pourquoi étudier les jeux ?

- Tâche bien structurée
- Mesure de performance facile
- Niveau abstrait
- Défi intellectuel
- Amusant
- Complexe
  - Tic-Tac-Toe : 39 états
  - Dames : 1040 états
  - Rubik's Cube : 1019 états
  - Echecs : 10120 états (facteur de branchement = 35)
  - Go

La théorie du jeu est l'étude des approches dans lesquelles des interactions stratégiques entre joueurs rationnels produisent des résultats en accord avec l'intérêt de ces joueurs mais correspondant souvent à un compromis.

Historique

- 1890 : roi et tour contre roi [Torres & Quevedo]
- 1950 : technique de base du MinMax [Shannon]
- 58-63 : Elagage alpha-beta [Mc Carthy, Newell & Simon]
- 1965 : recherche sur les méthodes des joueurs d'échecs [De Groot]
- ....
- 1992 : programme Chinook, vainqueur de l'US OPEN [schaefer]
- 1992 : programme de Backgammon Top 3 [Tesauro]
- 1997 : Deep Blue, vainqueur de Kasparov [Hsu, Campbell & al.]
- 2002 : récompense de $2,000,000 au GO
- 2011 : Watson, jeopardy

Caractéristiques

- Introduction de l'incertitude (toutes les actions ne sont pas contrôlées par le programme).
- Espace de recherche très vaste (il est donc impossible d'explorer tout l'espace pour trouver la meilleure action).
- Introduction d'une fonction d'évaluation statique (fonction de gain) et d'une stratégie de contrôle.

On se restreint généralement à un jeu sur plateau entre deux joueurs décrit par des états, des actions, un état initial et un but (pouvant être satisfait par plusieurs états).
Classification des jeux :

	Déterministe	Chance
info parfaite	Dames, Echecs, Go, Othello	Backgammon, Monopoly
info imparfaite	Master mind	Bridge, Poker, Scrabble

Fonction d'évaluation

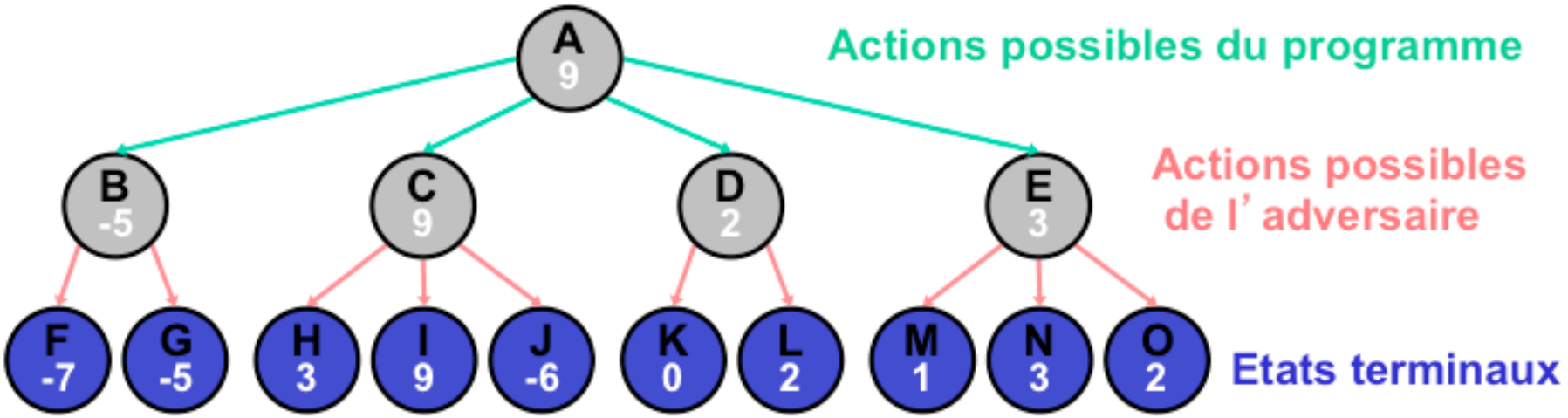
Une fonction d'évaluation statique estime la force d'une position (du point de vue du programme).
Sa valeur peut être négative et est indépendante des coups passés ou à venir à la différence de celle d'une heuristique.

$f(n) >> 0$: position favorable au programme et défavorable à l'adversaire.
$f(n) << 0$: position défavorable au programme et favorable à l'adversaire.
$f(n) = 0$: position équilibrée.
$f(n) = +\infty$: le programme gagne.
$f(n) = -\infty$: l'adversaire gagne.

Généralement il s'agit d'une fonction linéaire (somme pondérée des caractéristiques).
Ce que l'un des joueurs gagne l'autre le perd.

Pb de la recherche gourmande

Le programme jouera C
L'adversaire jouera J
Le programme à perdu !
Les états D et E auraient été préférables.



Le principe du MinMax

Le meilleur coup pour nous peut permettre à notre adversaire de gagner.
Nous ignorons les mouvements de l'adversaire.
Préparons-nous au pire (i.e. que l'adversaire jouera le meilleur coup).
Le programme choisit les actions qui maximisent la fonction d'évaluation en supposant que par la suite l'adversaire choisira les actions qui la minimiseront.

1. Etendre l'arbre de jeu jusqu'aux nœuds terminaux.
2. Calculer la valeur de la fonction de gain pour chaque nœuds terminaux.
3. Propager ces valeurs aux nœuds parents (non-terminaux) en affectant à un noeud :
   - la valeur minimum de ses fils lorsqu'ils résultent d'actions de l'adversaire
   - la valeur maximum de ses fils lorsqu'ils résultent d'actions du programme

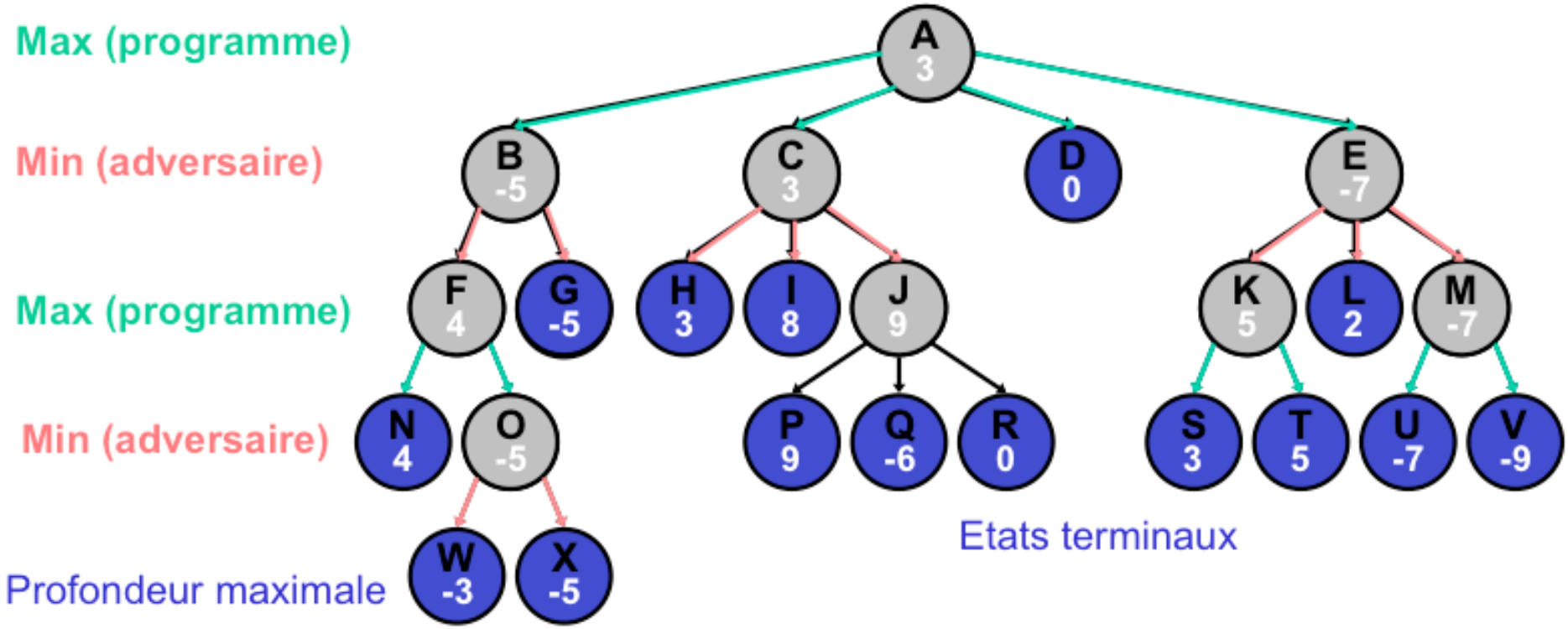
Dans le premier cas le nœud parent est dit MIN et MAX dans le second.

L'algorithme du MinMax

```
int minmax (Node s, int depth, int limit)
Vector v = new Vector();
si isTerminal(s) OR depth == limit alors
   return (GainEvaluation(s));
sinon
   // appliquer minmax sur les fils de s et mémoriser leurs valeurs
   tant que s.hasMoreSuccessors() faire
      v.addElement(minmax(s.getNextSuccessor(),depth+1,limit));
   fin tant que
   si isComputerTurn(s) alors
      return maxOf(v); // tour du programme
   sinon
      return minOf(v); // tour de l'adversaire
   fin si
fin si
```

limit correspond à la profondeur limite de recherche. Elle réduit l'explosion combinatoire en diminuant l'espace de recherche.

Exemple de MinMax



Principe de l'élagage alpha-beta

"Si vous pensez que c'est assurément mauvais, ne perdez pas de temps à vérifier à quel point c'est vraiment mauvais" Pat Winston

1. Etendre l'arbre de jeu jusqu'à une profondeur N par recherche en profondeur.
2. Ne plus développer les successeurs d'un nœud dès qu'il est évident que ce nœud ne sera pas choisi (compte tenu des nœuds déjà examinés)
   - Chaque nœud Max garde la trace d'une α-valeur égale à la valeur de son meilleur successeur trouvé jusqu'ici
   - Chaque nœud Min garde la trace d'une β-valeur égale à la valeur de son plus mauvais successeur trouvé jusqu'ici

Règles d'élagage

- Interrompre la recherche d'un nœud Max si sa α-valeur $\geq$ β-valeur de son parent
- Interrompre la recherche d'un nœud Min si sa β-valeur $\leq$ α-valeur de son parent

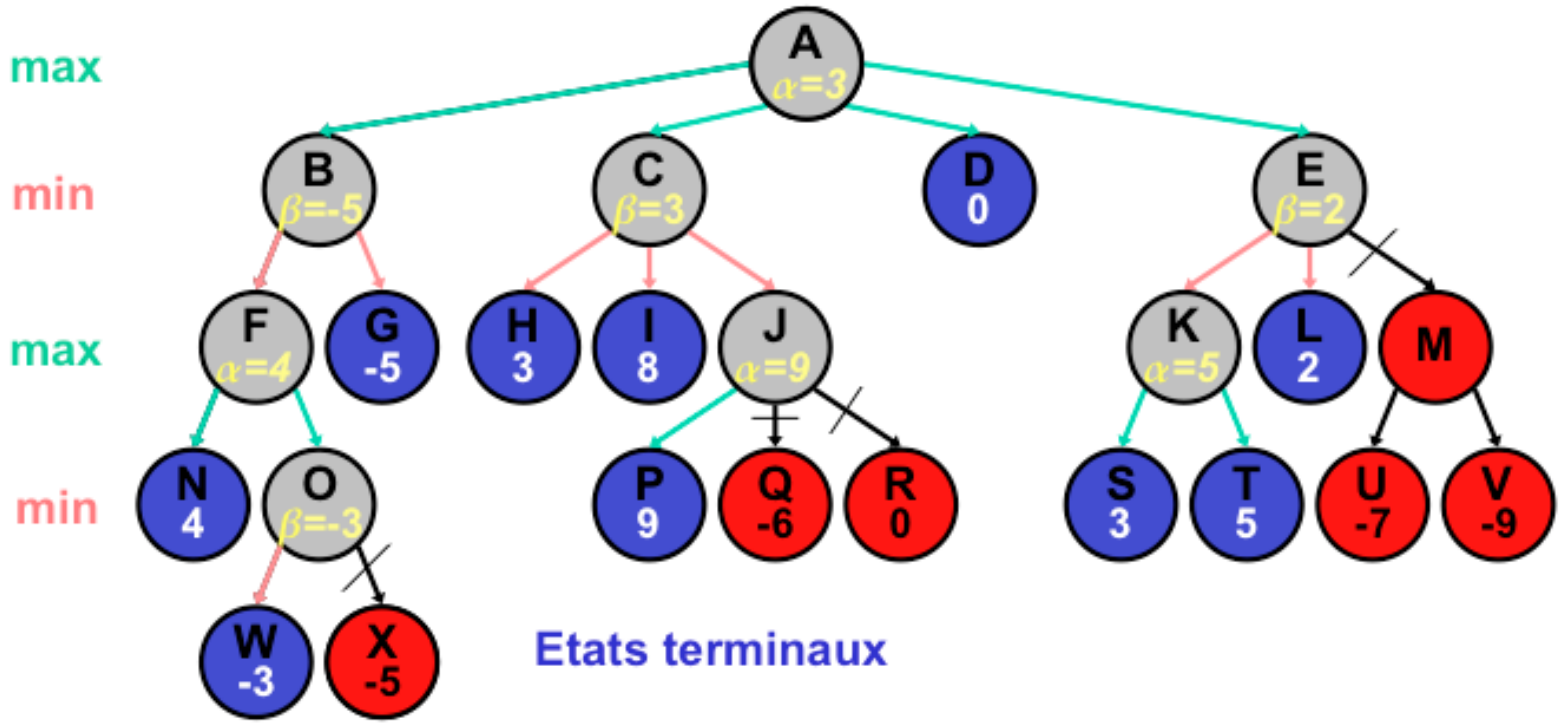
L'algorithme de l'élagage alpha-beta

Fct α-β (Node s, int alpha, int beta, int depth, int limit) //La valeur d'un nœud est encadrée par alpha et beta.
si isTerminal(s) OR depth == limit
    return (GainEvaluation(s));
sinon // appliquer alpha-beta sur les fils de s et mémoriser leurs valeurs
    s.alpha = -; s.beta = +;
    si (isComputerTurn(s))
        pour tout successeur si de s faire
            val = Fct α - β(si, max(alpha, s.alpha),beta, depth+1, limit)
            s.alpha = max(s.alpha, val)
            si s.alpha $\geq$ beta alors break
        return s.alpha
    sinon // tour de l'adversaire
        pout tour successeur si de s faire
            val = Fct α-β(si, alpha, min(beta, s.beta, depth+1, limit))
            s.beta = min(s.beta, val)

L'algorithme de l'élagage alpha-beta

```
α-β(Node s, int alpha, int beta, int limit)
si isTerminal(s) OR depth == limit alors
    return GainEvaluation(s)
sinon
    // appliquer alpha-beta sur les fils de s et mémoriser leurs valeurs
    s.alpha = -∞;
    s.beta = +∞;
    si isComputerTurn(s) alors
        pour tout successeur si de s faire
            val = α-β(si, max(alpha, s.alpha),beta)
            s.alpha = max(s.alpha, val)
            si s.alpha ≥ beta alors
                break
            fin si
        fin pour
        return s.alpha
    sinon
        pour tout successeur si de s faire faire
            val = α-β(si, alpha, min(beta, s.beta))
            s.beta = min(s.beta, val)
            si s.beta ≤ alpha alors
                break
            fin si
        fin pour
        return s.beta
    fin si
fin si
```

Exemple d'alpha-beta



Efficacité de l'alpha-beta

Dans le pire des cas :

- il n'y a pas d'élagage
- On examine b^d nœuds-feuilles

Dans le meilleur des cas :

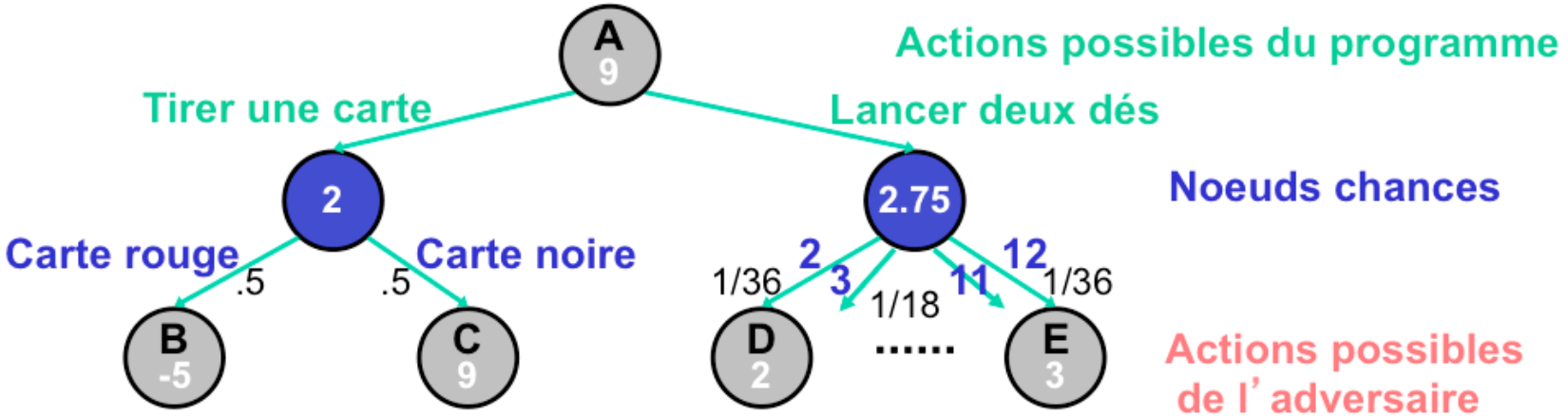
- le meilleur coup de chaque joueur est la première action étudiée
- On examine $2b^{d/2}$ nœuds-feuilles

Jeux non-déterministes

Certains jeux font intervenir la **chance** :

- lancé de dés
- tour une roue de la fortune
- distribution de cartes

Pour prendre en compte les éléments aléatoires, l'arbre de jeu est étendu pour incorporer des **nœuds probabilistes**.



En résumé

- **Les algorithmes de jeux** utilisent des arbres de recherche représentant alternativement les actions du programme et celles de l'adversaire.
- **La fonction d'évaluation** estime la qualité d'une situation pour chaque joueur :
  - une valeur positive correspond à une situation favorable au programme
  - une valeur négative correspond à une situation défavorable au programme
  - une valeur nulle correspond à un équilibre
- **MinMax** est une procédure qui choisit les actions en supposant que l'adversaire choisira toujours sa meilleure action.
- L'élagage **Alpha-Beta** est une procédure qui peut éliminer une grande partie de l'arbre de recherche permettant ainsi d'augmenter la profondeur de recherche.
- A l'origine les stratégies de jeux ont été étudiées pour aider la recherche en intelligence artificielle mais leur force réside principalement dans la puissance de calcul des machines.