Variante du minimax

Sylvain HUET

décembre 1994

1 Présentation

Le minimax est un algorithme employé principalement dans la programmation d'adversaire pour les jeux de réflexions tels que les échecs ou othello, ou d'une manière plus générale aux jeux vérifiant les critères suivants :

- les joueurs jouent successivement,
- dans une position donnée, l'éventail des coups jouables est fini.

Il s'en suit qu'à partir d'une position donnée, l'ensemble des coups possibles se schématise en une arborescence, où chaque noeud représente une position, et chacune de ses branches un coup jouable par le joueur qui a la main.

Le minimax utilise alors une fonction d'évaluation f capable d'attribuer à toute position un score exprimant en quelle mesure celle-ci est favorable ou défavorable à l'un des joueurs.

Pour une profondeur k, et pour une position P dans laquelle le joueur J a la main, l'algorithme du minimax détermine le coup qui assure à J, quels que soient les coups de l'adversaire, la meilleure évaluation possible k coups plus tard. Cela se fait en parcourant récursivement l'arbre de sommet P et de profondeur k de la manière suivante :

- si p est une feuille, retourner $f(p)$
- si p est un noeud où le joueur J a la main, évaluer récursivement tous les fils, et retourner la valeur max,
- si p est un noeud où l'adversaire a la main, évaluer récursivement tous les fils, et retourner la valeur min.

Cet algorithme parcourt la totalité de l'arbre, qui grossit de manière exponentielle en fonction de k, ce qui le rend rapidement inutilisable.

Une variante courante, l'alpha-bêta, utilise le même principe, mais effectue des coupes dans l'arbre.

La méthode qui va être ici développée envisage la recherche du meilleur coup de manière différente, tout en retournant exactement le même résultat, puisque la détermination du meilleur coup ne dépend que de la fonction d'évaluation f, et non de la technique de recherche.

Cette méthode est d'autant plus performante que la profondeur de recherche k et le nombre de coups jouables pour une position donnée sont grands.

2 Principe de la méthode

Idée directrice

Le principe premier de cette méthode consiste à remplacer la question "Quelle est l'évaluation maximale que je peux assurer dans k coups ?" par la question "Soit une valeur x de la fonction d'évaluation f, suis-je capable d'assurer ce score dans k coups ?".

La partie centrale de l'algorithme sera simplement capable de répondre à cette question, c'est-à-dire par oui ou par non (et si la réponse est oui, elle indiquera le coup qu'il faut jouer).

Pour connaître la meilleure évaluation possible dans k coups, il suffit alors de faire varier la valeur de x et de trouver la plus grande parmi celles qui obtiennent une réponse affirmative. Ceci peut se faire par une simple dichotomie, à condition (mais cela est habituellement vérifié) que l'ensemble des valeurs de f soit borné.

Coeur de la recherche

Etant donnée une valeur x de f, il s'agit de déterminer si k coups plus tard il est possible ou non au joueur J d'assurer l'évaluation x. On procèdera ainsi :

1. à partir de la position P, jouer k coups (peu importe lesquels), et évaluer la position Q ainsi obtenue : $y = f(Q)$,
2. si $y \geq x$, la position Q est favorable au joueur J, on change le dernier coup de l'adversaire ; si on a épuisé tous les coups, on remonte au coup précédent de l'adversaire (soit deux noeuds au-dessus); puis on redescend au niveau k en jouant des coups quelconques,

3. si $y < x$, la position Q est défavorable au joueur J, on change le dernier coup du joueur J ; si on a épuisé tous les coups, on remonte au coup précédent du joueur J (soit deux noeuds au-dessus); puis on redescend au niveau k en jouant des coups quelconques.

4. évaluer la nouvelle position Q : $y = f(x)$; recommencer en 1.

L'algorithme se termine lorsqu'il est impossible de remonter deux noeuds en amont, c'est-à-dire lorsque les coups du premier ou du second niveau de l'arbre sont épuisés : remonter de deux noeuds reviendrait alors à remonter en amont du sommet de l'arbre. Deux cas se présentent :

- les coups du premier niveau de l'arbre (le premier coup du joueur J) sont épuisés : il n'est pas possible d'assurer x,
- les coups du second niveau de l'arbre (la première réponse de l'adversaire au coup C du joueur J) sont épuisés : il est possible d'assurer x en jouant le coup C.

On peut écrire l'algorithme de la manière suivante :

```
fonction possible?(position P, profondeur k, evaluation x, joueur j)
/* j indique le joueur qui a la main sur la position P :
   joueur J ou adversaire */
. si (k=0)
         . y=f(P)
         . si (y>=x) retourne oui, sinon retourne non
. tant qu'il reste des coups c jouables pour le joueur j sur la position P
         . position Q = coup c jou\'e sur la position P
         . r = possible?(Q,k-1,x,joueur_suivant(j))
         . si (j=joueur J) et (r=oui) retourne oui
         . si (j=adversaire) et (r=non) retourne non
. si (j=joueur J), retourne non, sinon retourne oui
```

3 Analyse théorique

3.1 Particularisation du modèle

Pour l'analyse théorique, on considèrera un modèle de jeu particulier :

- chaque joueur joue l'un après l'autre (mais passer son tour est considéré comme un coup, ce qui permet toujours de se ramener à ce cas)
- à chaque tour, il y a un nombre n constant de coups jouables.

3.2 Calcul du nombre moyen de feuilles évaluées

Pour une valeur x de la fonction d'évaluation f, on se propose de calculer le nombre moyen N_n^k de feuilles évaluées pour une recherche de profondeur k.

Définition de p

Pour toute valeur x de la fonction f, et pour une profondeur k donnée, on définit p ainsi :
. si k est impair (le dernier coup de l'arbre est joué par le joeur J), **une feuille de l'arbre de jeu de profondeur k a la probabilité p d'avoir une évaluation strictement inférieure à x.**
. si k est pair, (le dernier coup de l'arbre est joué par l'adversaire) **une feuille de l'arbre de jeu de profondeur k a la probabilité p d'avoir une évaluation supérieure à x.**

Définition de $u_i(n,p)$

Soit i le nombre de coups restant à jouer pour atteindre la profondeur k désirée. Lorsque l'algorithme explore un coup parmi les n jouables, $u_i(n,p)$ est la probabilité que ce coup joué ne satisfasse pas son auteur. Satisfaire son auteur signifie :

- si le joueur J a la main, le coup permet d'assurer x en bas de l'arbre,
- si l'adversaire a la main, le coup permet d'empêcher x en bas de l'arbre.

Trivialement, la probabilité qu'aucun coup ne satisfasse son auteur, et qu'il faille donc revenir deux coups en arrière vaut : $q = u_i^n(n,p)$. Par ailleurs, pour qu'un coup ne satisfasse pas son auteur, il faut que le joueur suivant ait au moins un coup satisfaisant.

Il en ressort la relation de récurrence définissant la suite u :

$$\begin{cases} u_1(n,p) = p \\ u_{i+1}(n,p) = 1 - u_i^n(n,p) \end{cases}$$

Calcul de l'espérance du nombre de branches parcourues

On se place en un noeud donné de l'arbre, et on cherche à déterminer le nombre de branches moyen $v_i(n,p)$ qu'il faudra examiner.

On a vu que $u_i(n,p)$ représente la probabilité qu'un coup ne satisfasse pas son auteur, et qu'il soit donc nécessaire d'en essayer un autre. En tenant encore compte qu'il y a exactement n coups jouables, on obtient la définition suivante :

$$v_i(n,p) = \sum_{j=0}^{n-1} u_i^j(n,p) = \frac{1-u_i^n(n,p)}{1-u_i(n,p)}$$

Calcul de l'espérance totale du nombre de feuilles évaluées

On trouve immédiatement le nombre moyen de feuilles évaluées :

$$N_n^k(p) = \prod_{i=1}^{k} v_i(n,p)$$

Ceci se calcule facilement dans 3 cas :

- $p = 0$: $N_n^{2k}(0) = N_n^{2k+1}(0) = n^k$
- $p = 1$: $N_n^{2k-1}(1) = N_n^{2k}(1) = n^k$
- $p = 1 - p^n$: $N_n^k(p) = (\frac{p}{1-p})^k$

Pour n donné, on appelera p_n cette valeur remarquable de p : $p_n = 1 - p_n^n$.

4 Etude du comportement de $N_n^k(p)$

On ne s'intéressera qu'au cas $n \geq 2$.
On va montrer dans cette partie que pour k grand, la valeur $N_n^k(p_n)$ est équivalente au maximum de $N_n^k(p)$. Pour cela, on commencera par montrer que si $p \neq p_n$, une des sous-suites paires et impaires de u tend vers 0 et l'autre vers 1.

4.1 Comportement de u

Préliminaire : étude de $k_n(x) = x(1-x^n)$

Etudions le comportement de $k_n(x)$ sur $[0,1]$.

$$k_n'(x) = 1 - (n+1)x^n$$

$k_n'(x)$ possède une unique racine réelle x_n appartenant à $[0,1]$:

$$x_n = \sqrt[n]{\frac{1}{1+n}}$$

Or $k_n(0) = k_n(1) = 0$ et $k_n'(0) = 1$, il s'en suit que :

- $k_n(x) \geq 0$ sur $[0,1]$
- $k_n(x)$ strictement croissante sur $]0, x_n[$
- $k_n(x)$ strictement décroissante sur $]x_n, 1[$

Remarque importante

$$u_i(n,p) < p_n \Leftrightarrow u_{i+1}(n,p) > p_n$$
$$u_i(n,p) > p_n \Leftrightarrow u_{i+1}(n,p) < p_n$$

Etude de $u_{i+2} - u_i$

Examinons le signe de $u_{i+2}(n,p) - u_i(n,p) = 1 - (1 - u_i^n(n,p))^n - u_i(n,p)$. Pour cela considérons la fonction $h(x) = 1 - (1 - x^n)^n - x$ sur l'intervalle $[0,1]$.

$$h(x) = 1 - (1 - x^n)^n - x$$
$$h'(x) = n^2(x(1 - x^n))^{n-1} - 1$$
$$h'(x) = n^2 k_n^{n-1}(x) - 1$$

Du comportement de $k_n(x)$, on déduit celui de $h'(x)$:

- $h'(x)$ strictement croissante sur $]0, x_n[$
- $h'(x)$ strictement décroissante sur $]x_n, 1[$

$h'(x)$ a donc au maximum deux racines sur $[0,1]$, or :

- $h'(0) = h'(1) = -1$
- $h(0) = h(1) = h(p_n) = 0$

On en déduit :

- $h(x) < 0$ sur $]0, p_n[$
- $h(x) > 0$ sur $]p_n, 1[$

Ce qui signifie :

- $u_i(n,p) < p_n \Rightarrow u_{i+2}(n,p) < u_i(n,p)$
- $u_i(n,p) > p_n \Rightarrow u_{i+2}(n,p) > u_i(n,p)$

Or les suites $u_{2k}(n,p)$ et $u_{2k+1}(n,p)$ n'ont que trois points fixes : $0, p_n$ et 1. Donc si $p \neq p_n$, l'une des suites $u_{2k}(n,p)$ et $u_{2k+1}(n,p)$ tend vers 0 et l'autre vers 1.

4.2 Equivalent de $N_n^k(p)$

Etude de $v_i(n, p_n)$

Si $p = p_n$, la suite v est constante et vaut $\frac{p_n}{1-p_n}$.

Montrons : $(\frac{p_n}{1-p_n})^2 > n$
On a déjà : $n \geq 2 \Rightarrow p > \frac{1}{2}$

$$p = 1 - p^n \Rightarrow n = \frac{\ln(1-p)}{\ln p}$$

$$\text{Soit } A(p) = (\frac{p}{1-p})^2$$

$$\text{et } B(p) = \frac{\ln(1-p)}{\ln p}$$

Il faut montrer $A(p) > B(p)$. Or, $A(\frac{1}{2}) = B(\frac{1}{2}) = 1$.

$$A'(p) = 2\frac{p}{(1-p)^3} > 0$$

$$B'(p) = \frac{-p\ln p - (1-p)\ln(1-p)}{p(1-p)(\ln p)^2} > 0$$

Il suffit de montrer $\frac{A'(p)}{B'(p)} > 1$; on utilisera les formules élémentaires suivantes :

- $x > 0 \Rightarrow x \ln x \geq -\frac{1}{e}$
- $x \in]\frac{1}{2}, 1[\Rightarrow \ln x > 2(\ln \frac{1}{2})(1-x)$ (convexité)

Alors,

$$\frac{A'(p)}{B'(p)} = 2(\frac{p}{1-p})^2 . \frac{(\ln p)^2}{-p\ln p - (1-p)\ln(1-p)}$$
$$> 4e(\ln 2)^2 p^2$$
$$> e(\ln 2)^2 = 1,3.. > 1$$

Application au calcul de l'équivalent

Il s'agit de montrer :

$$\max_p N_n^k(p) \overset{k\to+\infty}{\sim} N_n^k(p_n) = (\frac{p_n}{1-p_n})^k$$

Soit $p \neq p_n$, alors, d'après l'étude de u, on a :

$$\forall \varepsilon > 0, \exists K, \forall k > K, v_i(n,p).v_{i+1}(n,p) < (1+\varepsilon)n$$

alors,

$$\forall k > 0, \frac{N_n^{K+2k}(p_n)}{N_n^{K+2k}(p)} = \prod_{i=1}^{K} \frac{v_i(n,p_n)}{v_i(n,p)} \cdot \prod_{i=1}^{k-1} \frac{v_{K+2i-1}(n,p_n).v_{K+2i}(n,p_n)}{v_{K+2i-1}(n,p).v_{K+2i}(n,p)}$$

$$\geq \prod_{i=1}^{K} \frac{v_i(n,p_n)}{v_i(n,p)} \cdot (\frac{1}{n(1+\varepsilon)} \cdot (\frac{p_n}{1-p_n})^2)^k$$

Ce qui diverge vers $+\infty$ en prenant ε assez petit, car $(\frac{1}{n} \cdot (\frac{p_n}{1-p_n})^2) > 1$ (d'après l'étude de $v_i(n,p_n)$).

5 Observations diverses

5.1 Minoration du nombre de feuilles évaluées

Soit x une valeur de f, supposons que toutes les feuilles de l'arbre de profondeur k ont une évaluation supérieure à x. Alors le nombre de feuilles évaluées sera : $N = n^{E(\frac{k}{2})}$, c'est-à-dire $N \sim (\sqrt{n})^k$.

De plus, pour toute position P et toute évaluation x données, il existe toujours un ordre de parcours des branches (du joueur J si x est possible, de l'adversaire si x n'est pas possible) qui permet d'effectuer le test en N évaluations.

Il en résulte que l'algorithme sera plus rapide si l'on examine les branches d'un noeud en commençant par le coup qui semble a priori le meilleur.

5.2 Limites du calcul statistique

Le calcul de $N_n^k(p)$ suppose une probabilité p homogène sur tous les sous-ensembles de l'arborescence. Or ceci n'est pas possible pour deux raisons :

- pour un jeu réel, certains coups mènent à des positions très favorables ou au contraire très défavorables quelque soit la suite du jeu
- pour les sous-ensembles de faibles profondeurs, le nombre de feuilles est trop faible pour respecter la probabilité globale.

Or, la déviation de $u_i(n,p)$ de p_n aboutit à un nombre moyen de branches visitées (pour chaque noeud) proche de $\sqrt{n}$. Il en résulte qu'en pratique, le nombre de feuilles visitées sera très inférieur à $(\frac{p_n}{1-p_n})^k$.

5.3 Discussion sur la signification de p_n

On a vu que lorsque l'on teste une évaluation x, on définit implicitement une valeur de p, qui représente la proportion de feuilles qui réalisent ou non (selon la parité de k) l'évaluation x.

De plus, $u_{k+1}(n,p)$ représente la probabilité de pouvoir assurer x. Or si $p \neq p_n$, $u_{k+1}(n,p)$ sera très proche de 0 ou de 1. Il s'en suit que p_n est une bonne approximation de la probabilité associée à la valeur x optimale. Et par suite que le nombre de feuilles évaluées au cours d'une étape de la dichotomie augmente lorsque la valeur y testée se rapproche de la valeur x optimale.

On vérifie expérimentalement ces résultats sur une arborescence aléatoire.

5.4 Observations sur la dichotomie

Il est à remarquer que si un coup c du joueur J dans la position P ne permet pas d'atteindre l'évaluation x, alors il ne permettra pas non plus d'atteindre une évaluation y supérieure à x.

Typiquement, on examine toujours dans le même ordre les branches d'un même noeud. Alors, si le i-ème coup du joueur J est le premier qui permet l'évaluation x, la prochaine étape de la dichotomie essayera de réaliser l'évaluation y avec $y > x$ en n'examinant les coups qu'à partir du i-ème.

Une observation plus "ludique" : rien n'oblige à poursuivre la dichotomie jusqu'au bout, on peut s'arrêter en cours et obtenir le coup qui assure une évaluation y proche de la meilleure évaluation x (on a alors toujours $y \leq x$, et on peut majorer la différence). Ceci permet de gagner du temps puisque le nombre de branches évaluées augmente lorsque la valeur y testée se rapproche de la valeur optimale x.