Algorithme MinMax – élagage AlphaBeta

1 Introduction

Ce document fait une présentation synthétique d'un algorithme standard, le *MinMax* ou encore *MiniMax*, et de son amélioration principale l'élagage AlphaBeta, permettant la sélection d'un coup dans les jeux dits « jeux à deux joueurs à somme nulle et information complète », famille dans laquelle se trouvent la plupart des jeux de réflexion (Othello, évidemment, mais aussi échecs, morpion, puissance 4, Go, etc) :

somme nulle : les gains d'un joueur sont exactement l'opposé des gains de l'autre joueur (ces gains peuvent donc être négatifs) ;

information complète : lors de sa prise de décision (i.e. du choix d'un coup à jouer dans le cas qui nous intéresse), chaque joueur connaît précisément
- ses possibilités d'action ;
- les possibilités d'action de son opposant (c'est-à-dire les répliques possibles de l'adversaire dans le cas d'Othello) ;
- les gains résultants de ces actions ;
- les motivations de son opposant.

MinMax est un algorithme, et plus généralement une stratégie, qui provient du domaine de la « Théorie des jeux » et a été identifié par Von Neumann il y a plus de 60 ans. Son amélioration la plus connue est l'algorithme « MinMax avec élagage AlphaBeta » qui est encore aujourd'hui à la base, dans des versions quelque peu améliorées, des programmes de jeu les plus évolués (Logistello pour Othello, DeeperBlue, DeepFritz pour les échecs, etc.).

Ce document se décompose de la manière suivante. Dans la section 2 on présente une version idéale de l'algorithme MinMax et on introduit la notion de recherche à profondeur donnée et la nécessité d'avoir une fonction d'évaluation. La section 3 fournit le pseudo-code de MinMax et discute quelques-unes de ses limites. Le principe de l'élagage AlphaBeta, qui améliore notablement l'algorithme MinMax, est décrit dans la section 4.

2 MinMax et fonction d'évaluation

2.1 Le MinMax idéal : recherche exhaustive

Rappelons tout d'abord que l'objectif des méthodes de recherche présentées dans ce document est de choisir un coup à jouer dans un jeu de réflexion. Ces méthodes seront donc utilisées pour que l'ordinateur puisse effectivement jouer à Othello.

Avant de considérer précisément le cas d'un ordinateur qui joue à Othello cependant, analysons un contexte de jeu humain contre humain (toujours pour le jeu Othello). Le principe de la recherche d'un coup par MinMax est un processus de réflexion très naturel que tout un chacun a déjà mis en œuvre. Etant donné une position, on suppose que c'est au joueur noir de jouer.

Dans la position donnée, Noir a une série de coups qu'il peut effectuer : pour chacun d'eux, il s'interroge sur les répliques éventuelles que peut faire Blanc, qui lui-même analyse pour chacune de ses répliques celles auxquelles peut procéder Noir, qui à son tour examine à nouveau l'ensemble des coups qu'il peut effectuer suite aux répliques de Blanc, etc. Ce processus de réflexion peut continuer ainsi et, dans le cas où les deux joueurs ont la capacité de mener cette réflexion jusqu'à ce qu'aucun des 2 joueurs ne puisse plus jouer alors il est facile pour Noir de décider du coup qu'il doit jouer. Il lui suffit en effet de choisir le coup qui, s'il existe, quelque soit la suite de répliques de Noir et Blanc imaginées mène à une victoire.

Ce processus de réflexion est généralement associé à un arbre de jeu tel que celui présenté sur la figure 1. Chaque nœud y correspond à une position de jeu et les branches correspondent aux différents coups que peut faire Noir ou Blanc à partir de cette position. Les feuilles sont les nœuds terminaux, desquels ne partent aucune branche, et correspondent dans la situation où une recherche exhaustive peut être conduite jusqu'à la fin de la partie. Des valeurs V ou D pour victoire ou défaite, du point de vue de Noir, sont indiquées au niveau des nœuds terminaux : ces valeurs précisent donc l'issue de la séquence de coups à partir du nœud racine (i.e. le sommet de l'arbre).

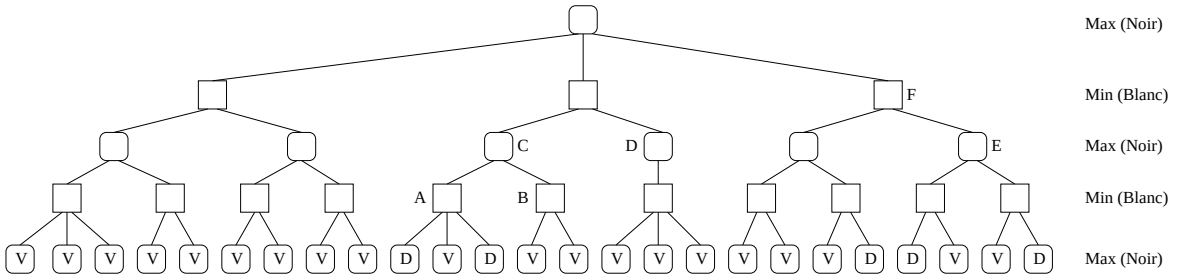

Fig. 1 – Arbre de recherche MinMax développé jusqu'à la fin de la partie. V ou D indique si la position terminale est une victoire de Noir ou bien une défaite. Un nœud Max correspond au cas où le joueur en question (ici, Noir) essaie de maximiser son gain alors qu'un nœud Min correspond au cas inverse.

L'arbre de la Figure 1 indique que Noir est gagnant de cette partie car il a la possibilité de jouer deux coups qui lui assureront de gagner quelles que soient les répliques de Blanc. Pour se rendre compte de cela, analysons brièvement l'arbre de jeu et indiquons pourquoi Noir est assuré de gagner.

Tout d'abord, pourquoi la stratégie de recherche prend le nom de MinMax ? Noir a 3 possibilités de jeu à partir de la position de départ. Il veut bien évidemment essayer de choisir le coup qui va MAXimiser ses gains, c'est-à-dire dans le cas présent de gagner la partie. Pour chacun des coups que peut répliquer Blanc, ce dernier veut à l'inverse MINimiser les gains de Noir (et donc, puisque c'est un jeu à somme nulle, maximiser ses gains). Ce qu'essaie ainsi de faire Noir est de maximiser le minimum des gains qu'il peut obtenir (puisque, encore une fois, Blanc fera tout pour lui faire avoir un gain minimum), d'où le nom MinMax.

Dans le cas de la Figure 1, Noir peut jouer indifféremment les deux premiers coups et est sûr de pouvoir gagner :

Premier coup. Le premier coup (branche la plus à gauche) mène à des situations terminales

qui sont toutes identifiées comme des victoires de Noir, il est donc évident que ce coup mène assurément à une victoire.

Deuxième coup. Dans le cas de la deuxième branche (celle du milieu en partant de la racine), la situation est un peu plus subtile car deux des nœuds terminaux, le premier et le troisième fils de A, sont étiquetés comme des défaites. Puisque Blanc a la possibilité en A de choisir l'un de ces deux coups il est évident qu'il ne s'en privera pas son objectif étant de minimiser le gain de Noir. Noir considère donc pour acquis que s'il arrive au nœud A, c'est-à-dire qu'il joue le premier coup possible lorsqu'il est à la position C, il perdra la partie : A peut être considéré comme un position de défaite. Néanmoins, B n'ayant que des fils étiquetés comme des victoires, Noir est sûr que s'il arrive dans la position B (deuxième fils de C) il gagnera la partie : on peut considérer que ce nœud est considéré comme une victoire. Ce qui a été vu permet donc de dire qu'en C, Noir a deux choix de coups, l'un menant à une victoire et l'autre à une défaite ; puisqu'il essaie de maximiser son gain, il choisira évidemment le coup qui l'amène à une position gagnante et C peut être considéré comme une position gagnante. La position D peut également être considérée comme victorieuse puisque tous les nœuds terminaux que l'on peut atteindre en descendant dans l'arbre à partir de D sont des victoires. Le second coup que peut faire Noir à partir de la racine est donc nécessairement un coup gagnant (si tant est que, au cours du jeu Noir ne fasse pas une "bourde" en C).

Troisième coup. S'agissant du troisème coup, il amène nécessairement à une position perdante. En effet, si l'on applique le même type de raisonnement que précédemment alors il s'avère que le nœud E correspond nécessairement à une défaite (Blanc se sera arrangé au niveau inférieur pour choisir le coup correspondant à une défaite) et donc la position en F est une défaite : Blanc a l'opportunité de choisir un coup qui mène à une défaite et il ne s'en privera pas.

Au final, la position à partir de laquelle est développé l'arbre est gagnante pour Max puisque Noir a le choix entre deux coups gagnants et un coup perdant et que, voulant maximiser son gain, il choisira l'un des coups gagnants. La Figure 2 illustre le réétiquetage des nœuds internes correspondant au raisonnement utilisé pour déterminer si la position est gagnante ou non : chaque nœud Max choisit le maximum de ses fils et inversement pour chaque nœud Min.

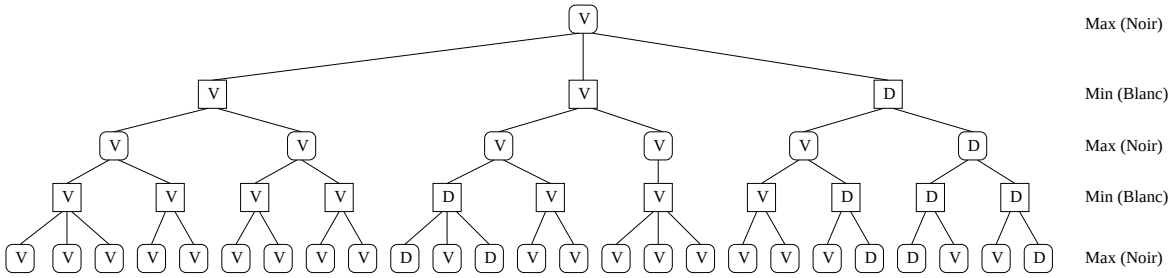

Fig. 2 – Détail de l'algorithme MinMax sur l'arbre de la Figure 1.

2.2 MinMax à profondeur donnée + fonction d'évaluation

Notons que cette situation de recherche jusqu'à l'atteinte de positions terminales ne peut exister que lorsque mise en œuvre à partir d'une position qui est proche de la fin de la partie. Il est facile d'imaginer que l'arbre de jeu ne peut être développé à partir de positions où le plateau de jeu n'est que peu rempli, et, *a fortiori*, à partir de la position de départ. (C'est en revanche possible de le faire à partir de la position de départ pour des jeux de réflexion très simples tels que le morpion, qui se joue sur un damier 3×3.) Pour le reste de la partie, il est donc nécessaire de mettre en œuvre la stratégie MinMax en utilisant une profondeur d'arbre prédéfinie et une fonction d'évaluation.

Cela signifie que le critère d'arrêt du développement de l'arbre de jeu est l'atteinte d'une hauteur maximum (la hauteur d'un arbre correspond à la longueur du plus long chemin qui y existe entre la racine de l'arbre et l'une de ses feuilles – l'arbre de la Figure 1 est de hauteur 4). Dans le cas d'arbres de jeu, on parle plus souvent de profondeur de recherche que de hauteur. Les feuilles de l'arbre ne sont donc pas des positions de fin de jeu et il n'est généralement pas possible de connaître de manière sûre et certaine l'étiquetage (V ou D) des positions se trouvant au niveau de feuilles. En d'autres termes, il n'est pas possible de dire si une feuille correspond à une position gagnante ou à une position perdante. Pour pallier ce problème, il est ainsi nécessaire de disposer d'une fonction d'évaluation, capable d'estimer le plus précisément possible la qualité d'une position, et qui servira à étiqueter les feuilles de l'arbre. Cet étiquetage consistera à l'affectation d'une valeur numérique à chacune des positions, valeur numérique calculée par la fonction d'évaluation. L'élaboration d'une fonction d'évaluation performante pour Othello fera l'objet d'un prochain document. La Figure 3 illustre l'application de MinMax lorsque les feuilles sont associées à une valeur numérique donnée par une fonction d'évaluation.

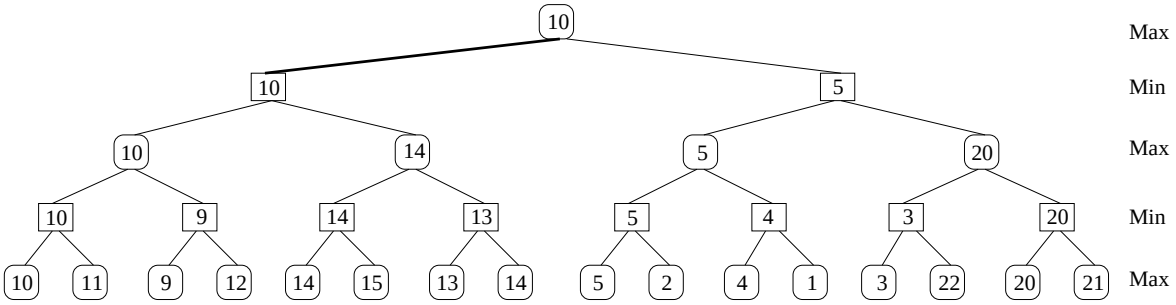

Fig. 3 – Application de l'algorithme MinMax en utilisant les notes obtenues par une fonction d'évaluation.

Dans ce cadre de travail, il est évident qu'une recherche dans un arbre à une profondeur plus grande permet généralement de choisir un coup de meilleur qualité que le coup retourné par un MinMax utilisant une profondeur plus faible.

Dans toute la suite du document, on considère la situation où une fonction d'évaluation est disponible et où les arbres développés le seront à une profondeur donnée.

```
int fonction minimax (int profondeur)
{
   if (game_over or profondeur = 0)
      return evaluation();

   int meilleur_score;
   move meilleur_coup;

   if (noeud == MAX) { //=Programme
      meilleur_score = -INFINITY;
      for (chaque coup possible  m) {
         jouer le coup m
         int score = minimax (profondeur - 1)
         annuler le coup m;
         if (score > meilleur_score) {
            meilleur_score = score;
            meilleur_coup = m ;
         }
      }
   }
   else { //type MIN = adversaire
      meilleur_coup = +INFINITY;
      for (chaque coup possible m) {
         jouer le coup m;
         int score = minimax (depth - 1)
         annuler le coup m;
         if (score < meilleur_score) {
            meilleur_score = score;
            meilleur_coup = m ;
         }
      }
   }
   return meilleur_coup ;
}
```

Tab. 1 – Pseudo-code de l'algorithme MinMax.

3 Algorithme MinMax

3.1 Pseudo-code

La Table 1 présente le pseudo-code correspondant à l'algorithme de recherche MinMax. Dans ce pseudo-code, on constate bien la distinction entre les nœuds Max et les nœuds Min : pour un type de nœud, on recherche le coup qui maximise les gain alors que pour l'autre type, on cherche le coup qui minimise le gain. Par ailleurs, l'aspect récursif de la fonction apparaît directement dans le code (pour éclairer ce point : cet aspect récursif provient directement du fait que le processus de réflexion mis en place à une profondeur n de l'arbre est exactement le même que celui mis en place à la profondeur $n - 2$).

Très peu de fonctions sont nécessaires : une fonction `eval()` qui correspond à la fonction d'évaluation, une fonction qui permet de générer tous les coups possibles à partir d'une position,

une fonction permettant de jouer un coup et une autre d'annuler un coup. Il faut noter que lors de la programmation proprement dite de la fonction MinMax, il est nécessaire de retourner non seulement le score mais en plus le meilleur coup (en général, on décide de passer une variable par adresse qui contiendra le meilleur coup).

3.2 Convention Negamax

Une version plus compacte de MinMax est présentée dans la Table 2. Cette version repose sur le constat du fait que minimiser une valeur revient à maximiser l'opposé de cette valeur. En utilisant cette remarque, il est possible d'éviter d'avoir à distinguer explicitement les nœuds Max et les nœuds Min. Notons que l'amélioration apportée par cette version n'est que de l'ordre de la programmation : du point de vue du coup choisi, l'utilisation de la convention Negamax de la Table 2 implémente exactement le même principe que le MinMax « classique ».

```
int negamax(int profondeur)
{
   if (game_over or profondeur <= 0)
      return eval();

   int meilleur_score = -INFINITY;
   move meilleur_coup ;
   for (chaque coup possible m) {
      jouer le coup m;
      int score = -negamax(profondeur - 1)
      annuler le coup m;
      if (score >= meilleur_score){
         meilleur_score = score ;
         meilleur_coup = m ;
      }
   }
   return meilleur_score;
}
```

Tab. 2 – Pseudo-code de l'algorithme MinMax en convention Negamax.

3.3 Limite de MinMax

La limite majeure de MinMax est que le nombre de branches que cet algorithme amène à développer est extrêmement important. En effet, si l'on considère que le facteur de branchement, c'est-à-dire le nombre moyen de coups potentiels à chaque position, est de 8 à Othello alors la recherche d'un coup en développant un arbre de profondeur d nécessite de générer de l'ordre de 8^d positions terminales. Cela implique donc que la fonction d'évaluation doit être appliquée à 8^d positions. Si l'on considère une fonction d'évaluation un peu évoluée qui prend 1×10^{-6} secondes à calculer le score d'une position et que l'on veut faire une recherche à une profondeur 10 (profondeur minimale utilisée par les meilleurs programme d'Othello) alors la recherche d'un coup par MinMax prend $8^{10} \times 10^{-6} = 1074$ secondes soit près de 18 minutes pour un seul coup. Précisons que les programmes comme Logistello génèrent des arbres de profondeur 14 à 18

(un joueur professionnel développe des arbres de profondeur similaire). L'algorithme sur lequel se développe celui utilisé par Logistello est une amélioration de MinMax qui permet d'éviter de développer des branches inutiles : cette amélioration appelée élagage AlphaBeta, introduit ce qui s'appelle des coupes dans l'arbre de recherche.

Cette amélioration est présentée dans la section suivante.

4 Elagage AlphaBeta

4.1 Principe

Le problème de MinMax est que les informations ne circulent que dans un seul sens : des feuilles vers la racine. Il est ainsi nécessaire d'avoir développé chaque feuille de l'arbre de recherche pour pouvoir propager les informations sur les scores des feuilles vers la racine.

Le principe de l'élagage AlphaBeta, que nous appellerons par abus de langage algorithme Alpha-Beta, est précisément d'éviter la génération de feuilles et de parties de l'arbre qui sont inutiles. Pour ce faire, cet algorithme repose sur l'idée de la génération de l'arbre selon un processus dit en « profondeur d'abord » où, avant de développer un frère d'un nœud (rappel : deux nœuds frères sont des nœuds qui ont le même parent), les fils sont développés. A cette idée vient se greffer la stratégie qui consiste à utiliser l'information en la remontant des feuilles et également en la redescendant vers d'autres feuilles.

Plus précisément, le principe de AlphaBeta est de tenir à jour deux variables α et β qui contiennent respectivement à chaque moment du développement de l'arbre la valeur minimale que le joueur peut espérer obtenir pour le coup à jouer étant donné la position où il se trouve et la valeur maximale. Certains développements de l'arbre sont arrêtés car ils indiquent qu'un des joueurs a l'opportunité de faire des coups qui violent le fait que α est obligatoirement la note la plus basse que le joueur Max sait pouvoir obtenir ou que β est la valeur maximale que le joueur Min autorisera Max à obtenir.

Un exemple concret est détaillé dans ce qui suit.

4.2 Pseudo-code

La Table 3 donne le pseudo-code de l'algorithme AlphaBeta (en convention Negamax). Cet algorithme est donc très similaire à MinMax, si ce n'est qu'il existe une petite condition de coupure qui s'active lorsqu'on viole l'une des conditions portant sur α ou β.

4.3 Illustration sur un exemple

Analysons une partie du déroulement de AlphaBeta sur l'arbre de la Figure 4, en supposant que l'arbre est normalement généré de la gauche vers la droite. Une coupure se produit en A car le développement de la partie gauche de A' permet de remonter l'information en A' que

```
int alphabeta(int profondeur, int alpha, int beta)
{
   if (game_over or profondeur <= 0)
      return eval();
   move meilleur_coup;
   for (chaque coup possible  m) {
      jouer le coup m;
      int score = -alphabeta(profondeur - 1, -beta, -alpha)
      annuler le coup m;
      if (score >= alpha){
         alpha = score ;
         meilleur_coup = m ;
         if (alpha >= beta)
            break;
      }
   }
   return alpha;
}
```

Tab. 3 – Algorithme AlphaBeta en convention Negamax. Cette function est appelée avec $\alpha = -\infty$ et $\beta = +\infty$.

$\alpha \geq 10$. Dès lors lorsque cette information est propagée vers le bas, et on trouve que Min (qui correspondait à Blanc, précédemment) a la possibilité d'obtenir la note de 9, qui est inférieure à α et il y a donc coupure de la branche A. Dans cette situation, on parle de coupe ou coupure alpha. L'information que $\beta \leq 10$ est alors remontée en B'. En appliquant l'algorithme décrit dans la table 3 pour la partie droite de B', nous informe qu'il est possible pour Max de jouer un coup lui apportera une note d'au moins 14, ce qui est supérieur à β, d'où la coupure de B. Il s'agit dans ce cas d'une coupe ou coupure beta. Les mêmes raisonnements permettent d'expliquer les autres coupures de l'arbre.

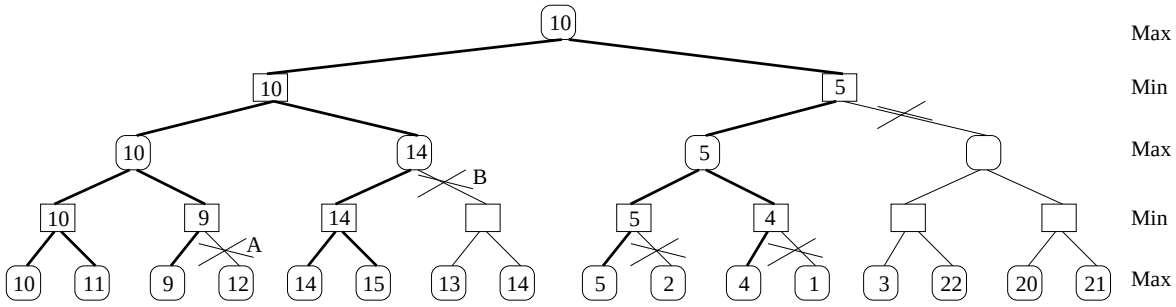

Fig. 4 – Algorithme AlphaBeta : les branches développées sont en trait gras ; les autres parties de l'arbre sont celles développées par MinMax mais pas par AlphaBeta.

4.4 Caractéristiques de AlphaBeta

Comme le montre l'exemple de la Figure 4, l'algorithme AlphaBeta permet la suppression de grandes parties de l'arbre complet. Il en résulte un gain de temps substantiel pour le choix d'un

coup à profondeur donnée. Plus précisément, en supposant un bon ordonnancement des coups étudiés, AlphaBeta permet d'explorer un nombre de nœuds qui est de l'ordre de la racine carrée du nombre de nœuds développés par MinMax. Ajoutons par ailleurs que la procédure AlphaBeta renvoie un coup qui a exactement la même valeur que MinMax.

4.5 Améliorations de AlphaBeta

AlphaBeta est la première étape de sophistication de MinMax. Plusieurs techniques peuvent s'y ajouter pour accélérer la recherche d'un coup. Pour cela, on pourra par exemple se référer aux trois références suivantes :
– le site de la Fédération Française d'Othello sur la programmation ;
– une très bonne description de T. Cazenave ;
– une page sur la programmation d'un jeu d'échecs.

5 Conclusion

Ce document décrit l'algorithme MinMax et la méthode d'élagage AlphaBeta, qui constituent la base du moteur d'intelligence artificielle de tous les jeux de réflexion du type échecs, Go, Othello, dames, etc. Le principe de recherche d'un coup suivant ces méthodes repose sur le développement d'un arbre de jeu dont les feuilles ont une note obtenue par l'utilisation d'une fonction d"évaluation. L'élaboration d'une fonction d'évaluation performante n'a pas été évoquée dans ce document mais elle constitue la seconde partie essentielle de la programmation d'un jeu de réflexion. Un prochain document sur le sujet sera bientôt mis à disposition.