Intelligence Artificielle
Jeux de Plateaux (2)

LAURENT SIMON
BORDEAUX-INP / LABRI
@lorensipro
lsimon@labri.fr

Résumé des épisodes précédents

L'α et l'ω de α-β

$\alpha\beta$
Élagage efficace de l'arbre

Élagage admissible

On veut trouver la même valeur d'évaluation finale du noeud racine sans développer tout l'arbre. Il faut donc élaguer des parties de l'arbre de recherche qui sont sans conséquence sur l'évaluation d'un noeud.

Intuitivement

Soit un noeud n dans l'arbre de recherche, tel que *Joueur* peut joueur en n.
S'il existe pour *Joueur* un choix m meilleur que n (soit à partir du noeud parent de n, soit plus haut dans l'arbre), n ne sera jamais effectivement joué.

$\alpha\beta$
Elagage efficace de l'arbre II

Deux types de coupes : α et β

- α : le meilleur choix à un instant donné pour Max sur le chemin développé. **La valeur α est croissante**
- β : le meilleur choix à un instant donné pour Min sur le chemin développé. **La valeur β est décroissante**

Les coupes auront lieu dès que α est supérieur à β

Algorithme $\alpha\beta$

La première fonction : *MaxValue*

Fonction $MaxValue(etat, \alpha, \beta)$ ▷ Évaluation niveau AMI
    $etat$: Plateau de jeu courant
    α : Meilleur évaluation courante pour AMI
    β : Meilleur évaluation courante pour ENNEMI

    Si $EstFeuille(etat)$ **Alors**
        Retourner $evalue(etat)$ ▷ Évaluation heuristique
    Fin Si
    Pour Tout successeur s de $etat$ **Faire**
        $\alpha \leftarrow max(\alpha, MinValue(s, \alpha, \beta))$
        Si $\alpha \geq \beta$ **Alors** ▷ Coupe β
            Retourner β
        Fin Si
    Fin Pour
    Retourner α
Fin Fonction

Algorithme $\alpha\beta$

La suite : *MinValue*

À bien noter

- Pour évaluer un plateau, on appelle *MaxValue* avec :
  plateau à évaluer, $\alpha = -\infty$ et $\beta = +\infty$
- Les variables α et β sont bien *locales*
- Elles sont changées par l'intermédiaire des valeurs de retour

Fonction $MinValue(etat, \alpha, \beta))$ ▷ Évaluation niveau ENNEMI
    $etat$: Plateau de jeu courant
    α : Meilleur évaluation courante pour AMI
    β : Meilleur évaluation courante pour ENNEMI

    Si $EstFeuille(etat)$ **Alors**
        Retourner $evalue(etat)$ ▷ Évaluation heuristique
    Fin Si
    Pour Tout successeur s de $etat$ **Faire**
        $\beta \leftarrow min(\beta, MaxValue(s, \alpha, \beta))$
        Si $\alpha \geq \beta$ **Alors** ▷ Coupe α
            Retourner α
        Fin Si
    Fin Pour
    Retourner β
Fin Fonction

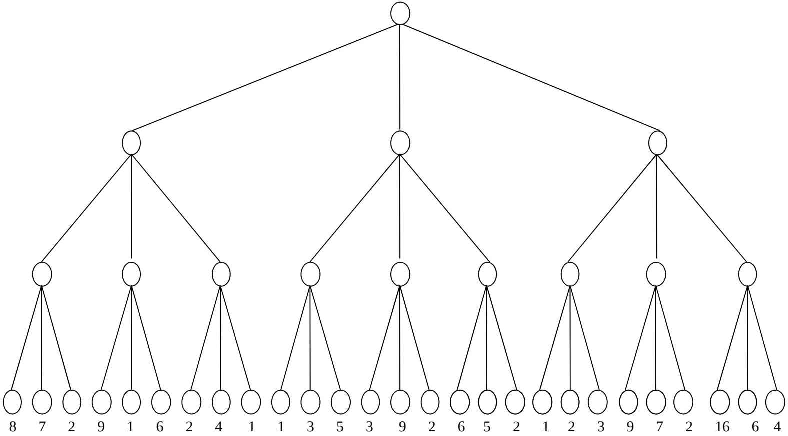
Algorithme $\alpha\beta$

Les deux fonctions ensembles

```
Fonction MaxValue(etat, α, β))                              ▷ Niveau AMI
    Si EstFeuille(etat) Alors
        Retourner evalue(etat)                              ▷ Évaluation heuristique
    Fin Si
    Pour Tout successeur s de etat Faire
        α ← max(α,MinValue(s, α, β))
        Si α ≥ β Alors                                      ▷ Coupe β
            Retourner β
        Fin Si
    Fin Pour
    Retourner α
Fin Fonction
```

```
Fonction MinValue(etat, α, β))
    Si EstFeuille(etat) Alors
        Retourner evalue(etat)                              ▷ Niveau ENNEMI
    Fin Si
    Pour Tout successeur s de etat Faire
        β ← min(β,MaxValue(s, α, β))
        Si α ≥ β Alors                                      ▷ Coupe α
            Retourner α
        Fin Si
    Fin Pour
    Retourner β
Fin Fonction
```

Exemple d'arbre de jeu



Propriétés de $\alpha\beta$

Efficacité théorique

Si on suppose que les fils sont ordonnées idéalement (ou presque), le coût de l'exploration est de $O(b^{d/2})$ au lieu de $O(b^d)$

- Le facteur de branchement théorique passe de b à $\sqrt{b}$
- La recherche peut aller deux fois plus loin dans l'arbre

Améliorations possibles

- Comment couper encore plus lors de la recherche ?
  - Recherche aspirante : α et β sont initialisés.
  - Si la valeur finale est bien dans $[\alpha, \beta]$ la recherche reste admissible
- ... D'autres encore à venir !

étude de α-β

Négamax

Une réécriture simple de MiniMax

Idée

Au lieu d'alterner deux fonctions Max/Min, on se restreint à une seule fonction en utilisant l'opposé du résultat à chaque niveau.

Attention

Cette famille d'algorithmes ne fonctionne que pour les niveaux pairs. Il faut adapter le résultat de la fonction heuristique sinon.

Négamax

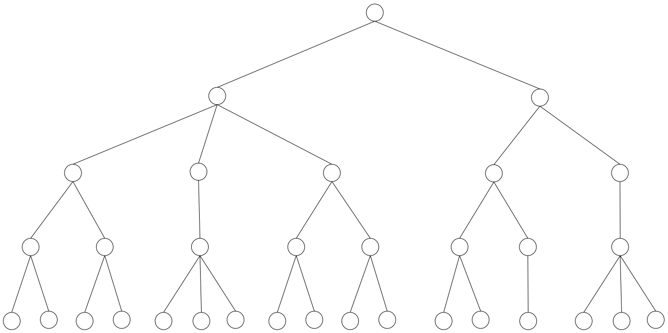
```
Fonction NegaMax(etat)                                  ▷ Évaluation niveau AMI
    etat : Plateau de jeu courant
    Meilleur : Évaluation du meilleur coup (localement)

    Si EstFeuille(etat) Alors                ▷ Fin de partie ou horizon atteint
        Retourner evalue(etat)                          ▷ Évaluation heuristique
    Fin Si
    Meilleur ← −∞
    Pour Tout successeur s de etat Faire
        val ← −NegaMax(s)
        Si val ≥ Meilleur Alors
            Meilleur ← val
        Fin Si
    Fin Pour
    Retourner Meilleur
Fin Fonction
```

Négamax appliqué à $\alpha\beta$

Fonction $Neg\alpha\beta(etat, \alpha, \beta)$ ▷ Évaluation niveau AMI

    Si $EstFeuille(etat)$ **Alors** ▷ Fin de partie ou horizon atteint
        Retourner $evalue(etat)$ ▷ Évaluation heuristique
    Fin Si
    Pour Tout successeur s de $etat$ **Faire**
        $val \leftarrow -Neg\alpha\beta(s, -\beta, -\alpha)$
        Si $val > \alpha$ **Alors**
            $\alpha \leftarrow val$
            Si $\alpha > \beta$ **Alors**
                Retourner α ▷ Coupe
            Fin Si
        Fin Si
    Fin Pour
    Retourner α
Fin Fonction

Exemple d'arbre de jeu avec Néga$\alpha\beta$



Quelles sont les performances de $\alpha\beta$?

Questions

- Combien de noeuds $\alpha\beta$ explore-t-il ?
- Quelle est la différence de performance avec MiniMax ?
- Avec le même temps donné quelle est la différence de profondeur entre les deux méthodes ?

Definition

On appelle arbre de jeu **uniforme de largeur** l, un arbre de jeu où tous les noeuds non terminaux ont exactement l fils.

MiniMax explore donc exactement l^p noeuds, où p est la profondeur de recherche.

Combien de noeuds $\alpha\beta$ explore-t-il ?

Quelles sont les performances de $\alpha\beta$?

Questions

- Combien de noeuds $\alpha\beta$ explore-t-il ?
- Quelle est la différence de performance avec MiniMax ?
- Avec le même temps donné quelle est la différence de profondeur entre les deux méthodes ?

Definition

On appelle arbre de jeu **uniforme de largeur** l, un arbre de jeu où tous les noeuds non terminaux ont exactement l fils.

MiniMax explore donc exactement l^p noeuds, où p est la profondeur de recherche.

Combien de noeuds $\alpha\beta$ explore-t-il ?

Quelles sont les performances de $\alpha\beta$?

Questions

- Combien de noeuds $\alpha\beta$ explore-t-il ?
- Quelle est la différence de performance avec MiniMax ?
- Avec le même temps donné quelle est la différence de profondeur entre les deux méthodes ?

Definition

On appelle arbre de jeu **uniforme de largeur** l, un arbre de jeu où tous les noeuds non terminaux ont exactement l fils.

MiniMax explore donc exactement l^p noeuds, où p est la profondeur de recherche.

Combien de noeuds $\alpha\beta$ explore-t-il ?

Quelles sont les performances de $\alpha\beta$?

Questions

- Combien de noeuds $\alpha\beta$ explore-t-il ?
- Quelle est la différence de performance avec MiniMax ?
- Avec le même temps donné quelle est la différence de profondeur entre les deux méthodes ?

Definition

On appelle arbre de jeu **uniforme de largeur** l, un arbre de jeu où tous les noeuds non terminaux ont exactement l fils.

MiniMax explore donc exactement l^p noeuds, où p est la profondeur de recherche.
Combien de noeuds $\alpha\beta$ explore-t-il ?

Types de noeuds visités lors de l'exploration

Tous les noeuds élagués lors de la recherche $Neg\alpha\beta$ le sont dès que $\alpha \geq \beta$. L'appel récursif $Neg\alpha\beta$ se ferait sur la fenêtre $[\alpha, \beta]$ avec $\alpha \geq \beta$.

Trois types de noeuds jamais élagués

À chaque niveau de l'arbre, $\alpha\beta$ est appelé avec une certaine fenêtre d'appel $[\alpha, \beta]$. Trois types de noeuds ne *peuvent donc* jamais être élagués.

1. la fenêtre d'appel est $[-\infty, +\infty]$
2. la fenêtre d'appel est $[-\infty, b]$ avec $b \neq +\infty$
3. la fenêtre d'appel est $[a, +\infty]$ avec $a \neq -\infty$

Note 1 : c'est en supposant que ∞ n'est pas une valeur heuristique.
Note 2 : les fils d'un noeud non élagué peuvent bien entendu l'être.

Types de noeuds visités lors de l'exploration

Tous les noeuds élagués lors de la recherche $Neg\alpha\beta$ le sont dès que $\alpha \geq \beta$. L'appel récursif $Neg\alpha\beta$ se ferait sur la fenêtre $[\alpha, \beta]$ avec $\alpha \geq \beta$.

Trois types de noeuds jamais élagués

À chaque niveau de l'arbre, $\alpha\beta$ est appelé avec une certaine fenêtre d'appel $[\alpha, \beta]$. Trois types de noeuds ne *peuvent donc* jamais être élagués.

1. la fenêtre d'appel est $[-\infty, +\infty]$
2. la fenêtre d'appel est $[-\infty, b]$ avec $b \neq +\infty$
3. la fenêtre d'appel est $[a, +\infty]$ avec $a \neq -\infty$

Note 1 : c'est en supposant que ∞ n'est pas une valeur heuristique.
Note 2 : les fils d'un noeud non élagué peuvent bien entendu l'être.

Types de noeuds visités lors de l'exploration

Tous les noeuds élagués lors de la recherche $Neg\alpha\beta$ le sont dès que $\alpha \geq \beta$. L'appel récursif $Neg\alpha\beta$ se ferait sur la fenêtre $[\alpha, \beta]$ avec $\alpha \geq \beta$.

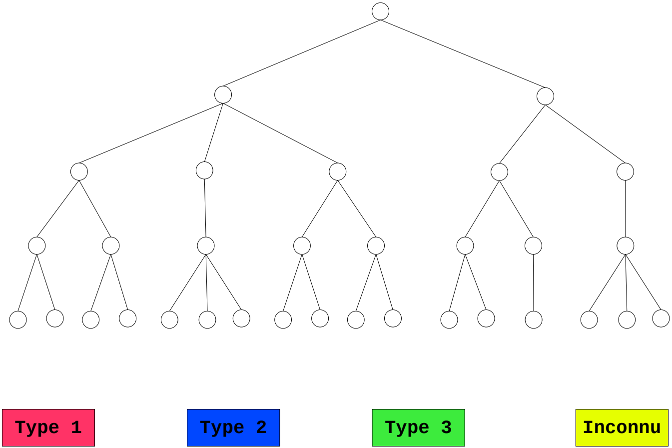
Trois types de noeuds jamais élagués

À chaque niveau de l'arbre, $\alpha\beta$ est appelé avec une certaine fenêtre d'appel $[\alpha, \beta]$. Trois types de noeuds ne *peuvent donc* jamais être élagués.

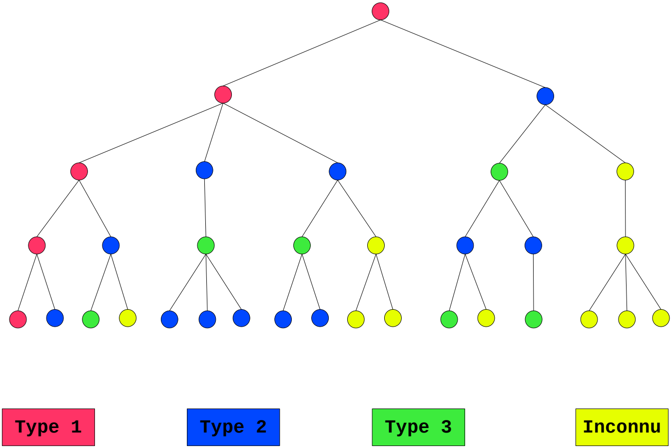
1. la fenêtre d'appel est $[-\infty, +\infty]$
2. la fenêtre d'appel est $[-\infty, b]$ avec $b \neq +\infty$
3. la fenêtre d'appel est $[a, +\infty]$ avec $a \neq -\infty$

Note 1 : c'est en supposant que ∞ n'est pas une valeur heuristique.
Note 2 : les fils d'un noeud non élagué peuvent bien entendu l'être.

Types de noeuds : exemple



Types de noeuds : exemple



Types de noeuds visités par $\alpha\beta$

Visite de l'arbre critique

L'algorithme $\alpha\beta$ utilisé avec la fenêtre $[-\infty, +\infty]$ regarde au moins *l'arbre critique*, c'est-à-dire l'ensemble des noeuds de types 1, 2 et 3, et uniquement celui-ci dans le cas où l'arbre est parfaitement ordonné.

Si l'arbre n'est pas parfaitement ordonné, on visitera plus de noeuds, jusqu'à l'arbre complet (l^p noeuds).
Quelle est la borne inférieur ?

Types de noeuds visités par $\alpha\beta$

Visite de l'arbre critique

L'algorithme $\alpha\beta$ utilisé avec la fenêtre $[-\infty, +\infty]$ regarde au moins *l'arbre critique*, c'est-à-dire l'ensemble des noeuds de types 1, 2 et 3, et uniquement celui-ci dans le cas où l'arbre est parfaitement ordonné.

Si l'arbre n'est pas parfaitement ordonné, on visitera plus de noeuds, jusqu'à l'arbre complet (l^p noeuds).
Quelle est la borne inférieur ?

Types de noeuds visités par $\alpha\beta$

Visite de l'arbre critique

L'algorithme $\alpha\beta$ utilisé avec la fenêtre $[-\infty, +\infty]$ regarde au moins *l'arbre critique*, c'est-à-dire l'ensemble des noeuds de types 1, 2 et 3, et uniquement celui-ci dans le cas où l'arbre est parfaitement ordonné.

Si l'arbre n'est pas parfaitement ordonné, on visitera plus de noeuds, jusqu'à l'arbre complet (l^p noeuds).
Quelle est la borne inférieur ?

Types de noeuds visités par $\alpha\beta$

Visite de l'arbre critique

L'algorithme $\alpha\beta$ utilisé avec la fenêtre $[-\infty, +\infty]$ regarde au moins *l'arbre critique*, c'est-à-dire l'ensemble des noeuds de types 1, 2 et 3, et uniquement celui-ci dans le cas où l'arbre est parfaitement ordonné.

Si l'arbre n'est pas parfaitement ordonné, on visitera plus de noeuds, jusqu'à l'arbre complet (l^p noeuds).
Quelle est la borne inférieur ?

Comptage des différents noeuds

Types de noeuds :

- type 1 : racine et fils ainés des type 1
- type 2 : cadets de type 1 et tous les fils de type 3
- type 3 : fils aînés des type 2
- inconnus : cadets des type 2 et les fils des inconnus

Preuve par récurrence

Soit u_n le nombre de feuilles de type 2,
v_n le nombre de feuilles de type 2 pour un arbre critique uniforme de largeur l et de profondeur n.

Suppositions :

$u_0 = 0,\ v_0 = 0$
$\forall n > 0,\ u_n = lv_{n-1} + l - 1,\ v_n = u_{n-1}$

Démonstration :

$\Rightarrow \forall n > 2,\ u_n = lu_{n-2} + l - 1$
$\Rightarrow \forall n > 2,\ u_n + 1 = l(u_{n-2} + 1)$
$\Rightarrow \forall n > 2,\ u_n = \begin{cases} l^{n/2} - 1 & \text{Si } n \text{ est pair} \\ l^{(n-1)/2}l - 1 & \text{Si } n \text{ est impair} \end{cases}$
$\Rightarrow \forall n,\ u_n = l^{n/2} - 1$

Comptage des différents noeuds

Types de noeuds :

- type 1 : racine et fils ainés des type 1
- type 2 : cadets de type 1 et tous les fils de type 3
- type 3 : fils aînés des type 2
- inconnus : cadets des type 2 et les fils des inconnus

Preuve par récurrence

Soit u_n le nombre de feuilles de type 2,
v_n le nombre de feuilles de type 2 pour un arbre critique uniforme de largeur l et de profondeur n.

Suppositions :

$u_0 = 0, \; v_0 = 0$

$\forall n > 0, \; u_n = l v_{n-1} + l - 1, \; v_n = u_{n-1}$

Démonstration :

$\Rightarrow \forall n > 2, \; u_n = l u_{n-2} + l - 1$

$\Rightarrow \forall n > 2, \; u_n + 1 = l(u_{n-2} + 1)$

$$\Rightarrow \forall n > 2, \; u_n = \begin{cases} l^{n/2} - 1 & \text{Si } n \text{ est pair} \\ l^{(n-1)/2} l - 1 & \text{Si } n \text{ est impair} \end{cases}$$

$\Rightarrow \forall n, \; u_n = l^{n/2} - 1$

Comptage des différents noeuds

Types de noeuds :

- type 1 : racine et fils ainés des type 1
- type 2 : cadets de type 1 et tous les fils de type 3
- type 3 : fils aînés des type 2
- inconnus : cadets des type 2 et les fils des inconnus

Preuve par récurrence
Soit u_n le nombre de feuilles de type 2,
v_n le nombre de feuilles de type 2 pour un arbre critique uniforme de largeur l et de profondeur n.
Suppositions :
$u_0 = 0,\ v_0 = 0$
$\forall n > 0,\ u_n = lv_{n-1} + l - 1,\ v_n = u_{n-1}$
Démonstration :
$\Rightarrow \forall n > 2,\ u_n = lu_{n-2} + l - 1$
$\Rightarrow \forall n > 2,\ u_n + 1 = l(u_{n-2} + 1)$

$$\Rightarrow \forall n > 2,\ u_n = \begin{cases} l^{n/2} - 1 & \text{Si } n \text{ est pair} \\ l^{(n-1)/2}l - 1 & \text{Si } n \text{ est impair} \end{cases}$$

$\Rightarrow \forall n,\ u_n = l^{n/2} - 1$

Comptage des différents noeuds

Types de noeuds :

- type 1 : racine et fils ainés des type 1
- type 2 : cadets de type 1 et tous les fils de type 3
- type 3 : fils aînés des type 2
- inconnus : cadets des type 2 et les fils des inconnus

Preuve par récurrence
Soit u_n le nombre de feuilles de type 2,
v_n le nombre de feuilles de type 2 pour un arbre critique uniforme de largeur l et de profondeur n.
Suppositions :

$u_0 = 0,\ v_0 = 0$
$\forall n > 0,\ u_n = lv_{n-1} + l - 1,\ v_n = u_{n-1}$

Démonstration :

$\Rightarrow \forall n > 2,\ u_n = lu_{n-2} + l - 1$
$\Rightarrow \forall n > 2,\ u_n + 1 = l(u_{n-2} + 1)$
$$\Rightarrow \forall n > 2,\ u_n = \begin{cases} l^{n/2} - 1 & \text{Si } n \text{ est pair} \\ l^{(n-1)/2} l - 1 & \text{Si } n \text{ est impair} \end{cases}$$
$\Rightarrow \forall n,\ u_n = l^{n/2} - 1$

Comptage des différents noeuds

Types de noeuds :

- type 1 : racine et fils ainés des type 1
- type 2 : cadets de type 1 et tous les fils de type 3
- type 3 : fils aînés des type 2
- inconnus : cadets des type 2 et les fils des inconnus

Preuve par récurrence
Soit u_n le nombre de feuilles de type 2,
v_n le nombre de feuilles de type 2 pour un arbre critique uniforme de largeur l et de profondeur n.
Suppositions :

$u_0 = 0,\ v_0 = 0$
$\forall n > 0,\ u_n = lv_{n-1} + l - 1,\ v_n = u_{n-1}$

Démonstration :

$\Rightarrow \forall n > 2,\ u_n = lu_{n-2} + l - 1$
$\Rightarrow \forall n > 2,\ u_n + 1 = l(u_{n-2} + 1)$
$$\Rightarrow \forall n > 2,\ u_n = \begin{cases} l^{n/2} - 1 & \text{Si } n \text{ est pair} \\ l^{(n-1)/2}l - 1 & \text{Si } n \text{ est impair} \end{cases}$$
$\Rightarrow \forall n,\ u_n = l^{n/2} - 1$

Comptage des différents noeuds

Types de noeuds :

- type 1 : racine et fils ainés des type 1
- type 2 : cadets de type 1 et tous les fils de type 3
- type 3 : fils aînés des type 2
- inconnus : cadets des type 2 et les fils des inconnus

Preuve par récurrence
Soit u_n le nombre de feuilles de type 2,
v_n le nombre de feuilles de type 2 pour un arbre critique uniforme de largeur l et de profondeur n.
Suppositions :

$$u_0 = 0,\ v_0 = 0$$
$$\forall n > 0,\ u_n = lv_{n-1} + l - 1,\ v_n = u_{n-1}$$

Démonstration :

$$\Rightarrow \forall n > 2,\ u_n = lu_{n-2} + l - 1$$
$$\Rightarrow \forall n > 2,\ u_n + 1 = l(u_{n-2} + 1)$$
$$\Rightarrow \forall n > 2,\ u_n = \begin{cases} l^{n/2} - 1 & \text{Si } n \text{ est pair} \\ l^{(n-1)/2}l - 1 & \text{Si } n \text{ est impair} \end{cases}$$
$$\Rightarrow \forall n,\ u_n = l^{n/2} - 1$$

Comptage des différents noeuds

Types de noeuds :

- type 1 : racine et fils ainés des type 1
- type 2 : cadets de type 1 et tous les fils de type 3
- type 3 : fils aînés des type 2
- inconnus : cadets des type 2 et les fils des inconnus

Preuve par récurrence
Soit u_n le nombre de feuilles de type 2,
v_n le nombre de feuilles de type 2 pour un arbre critique uniforme de largeur l et de profondeur n.
Suppositions :

$u_0 = 0,\ v_0 = 0$
$\forall n > 0,\ u_n = lv_{n-1} + l - 1,\ v_n = u_{n-1}$

Démonstration :

$\Rightarrow \forall n > 2,\ u_n = lu_{n-2} + l - 1$
$\Rightarrow \forall n > 2,\ u_n + 1 = l(u_{n-2} + 1)$

$$\Rightarrow \forall n > 2,\ u_n = \begin{cases} l^{n/2} - 1 & \text{Si } n \text{ est pair} \\ l^{(n-1)/2}l - 1 & \text{Si } n \text{ est impair} \end{cases}$$

$\Rightarrow \forall n,\ u_n = l^{n/2} - 1$

Comptage des différents noeuds

Types de noeuds :

- type 1 : racine et fils ainés des type 1
- type 2 : cadets de type 1 et tous les fils de type 3
- type 3 : fils aînés des type 2
- inconnus : cadets des type 2 et les fils des inconnus

Preuve par récurrence
Soit u_n le nombre de feuilles de type 2,
v_n le nombre de feuilles de type 2 pour un arbre critique uniforme de largeur l et de profondeur n.
Suppositions :

$u_0 = 0,\ v_0 = 0$
$\forall n > 0,\ u_n = lv_{n-1} + l - 1,\ v_n = u_{n-1}$

Démonstration :

$\Rightarrow \forall n > 2,\ u_n = lu_{n-2} + l - 1$
$\Rightarrow \forall n > 2,\ u_n + 1 = l(u_{n-2} + 1)$
$$\Rightarrow \forall n > 2,\ u_n = \begin{cases} l^{n/2} - 1 & \text{Si } n \text{ est pair} \\ l^{(n-1)/2}l - 1 & \text{Si } n \text{ est impair} \end{cases}$$
$\Rightarrow \forall n,\ u_n = l^{n/2} - 1$

Comptage des différents noeuds

Types de noeuds :

- type 1 : racine et fils ainés des type 1
- type 2 : cadets de type 1 et tous les fils de type 3
- type 3 : fils aînés des type 2
- inconnus : cadets des type 2 et les fils des inconnus

Preuve par récurrence
Soit u_n le nombre de feuilles de type 2,
v_n le nombre de feuilles de type 2 pour un arbre critique uniforme de largeur l et de profondeur n.
Suppositions :

$u_0 = 0,\ v_0 = 0$
$\forall n > 0,\ u_n = l v_{n-1} + l - 1,\ v_n = u_{n-1}$

Démonstration :

$\Rightarrow \forall n > 2,\ u_n = l u_{n-2} + l - 1$
$\Rightarrow \forall n > 2,\ u_n + 1 = l(u_{n-2} + 1)$
$$\Rightarrow \forall n > 2,\ u_n = \begin{cases} l^{n/2} - 1 & \text{Si } n \text{ est pair} \\ l^{(n-1)/2} l - 1 & \text{Si } n \text{ est impair} \end{cases}$$
$\Rightarrow \forall n,\ u_n = l^{n/2} - 1$

Comptage des différents noeuds

Types de noeuds :

- type 1 : racine et fils ainés des type 1
- type 2 : cadets de type 1 et tous les fils de type 3
- type 3 : fils aînés des type 2
- inconnus : cadets des type 2 et les fils des inconnus

En résumé

On a donc compté

- $l^{n/2} - 1$ noeuds de type 2
- d'où $l^{(n-1)/2} - 1$ noeuds de type 3
- et 1 noeud de type 1

Soit, au mieux pour $\alpha\beta$ de l'ordre de :

$$l^{n/2} - 1 + l^{(n-1)/2} - 1 + 1$$

ou encore

$$l^{n/2}$$

noeuds

Comptage des différents noeuds

Types de noeuds :

- type 1 : racine et fils ainés des type 1
- type 2 : cadets de type 1 et tous les fils de type 3
- type 3 : fils aînés des type 2
- inconnus : cadets des type 2 et les fils des inconnus

En résumé

On a donc compté

- $l^{n/2} - 1$ noeuds de type 2
- d'où $l^{(n-1)/2} - 1$ noeuds de type 3
- et 1 noeud de type 1

Soit, au mieux pour $\alpha\beta$ de l'ordre de :

$$l^{n/2} - 1 + l^{(n-1)/2} - 1 + 1$$

ou encore

$$l^{n/2}$$

noeuds

Comptage des différents noeuds

Types de noeuds :

- type 1 : racine et fils ainés des type 1
- type 2 : cadets de type 1 et tous les fils de type 3
- type 3 : fils aînés des type 2
- inconnus : cadets des type 2 et les fils des inconnus

En résumé

On a donc compté

- $l^{n/2} - 1$ noeuds de type 2
- d'où $l^{(n-1)/2} - 1$ noeuds de type 3
- et 1 noeud de type 1

Soit, au mieux pour $\alpha\beta$ de l'ordre de :

$$l^{n/2} - 1 + l^{(n-1)/2} - 1 + 1$$

ou encore

$$l^{n/2}$$

noeuds

Comptage des différents noeuds

Types de noeuds :

- type 1 : racine et fils ainés des type 1
- type 2 : cadets de type 1 et tous les fils de type 3
- type 3 : fils aînés des type 2
- inconnus : cadets des type 2 et les fils des inconnus

En résumé

On a donc compté

- $l^{n/2} - 1$ noeuds de type 2
- d'où $l^{(n-1)/2} - 1$ noeuds de type 3
- et 1 noeud de type 1

Soit, au mieux pour $\alpha\beta$ de l'ordre de :

$$l^{n/2} - 1 + l^{(n-1)/2} - 1 + 1$$

ou encore

$$l^{n/2}$$

noeuds

Comptage des différents noeuds

Types de noeuds :

- type 1 : racine et fils ainés des type 1
- type 2 : cadets de type 1 et tous les fils de type 3
- type 3 : fils aînés des type 2
- inconnus : cadets des type 2 et les fils des inconnus

En résumé

On a donc compté

- $l^{n/2} - 1$ noeuds de type 2
- d'où $l^{(n-1)/2} - 1$ noeuds de type 3
- et 1 noeud de type 1

Soit, au mieux pour $\alpha\beta$ de l'ordre de :

$$l^{n/2} - 1 + l^{(n-1)/2} - 1 + 1$$

ou encore

$$l^{n/2}$$

noeuds

Résultats des performances

Encadrement des performances de $\alpha\beta$

Le nombre de feuilles évaluées par $\alpha\beta$ est compris entre $l^{p/2}$ et l^p pour un arbre de largeur l et de profondeur p

Nombre de noeuds visités

$\alpha\beta$ visite au minimum un nombre de noeuds de l'ordre de $l^{p/2}$

Comparaison MiniMax / $\alpha\beta$

Au pire, $\alpha\beta$ explore tous les noeuds de l'arbre. Au mieux, $\alpha\beta$ peut voir à un horizon deux fois plus lointain que MiniMax dans le même temps.

L'ordre de développement des fils d'un noeud est primordial pour obtenir de bonnes performances !

Résultats des performances

Encadrement des performances de $\alpha\beta$

Le nombre de feuilles évaluées par $\alpha\beta$ est compris entre $l^{p/2}$ et l^p pour un arbre de largeur l et de profondeur p

Nombre de noeuds visités

$\alpha\beta$ visite au minimum un nombre de noeuds de l'ordre de $l^{p/2}$

Comparaison MiniMax / $\alpha\beta$

Au pire, $\alpha\beta$ explore tous les noeuds de l'arbre. Au mieux, $\alpha\beta$ peut voir à un horizon deux fois plus lointain que MiniMax dans le même temps.

L'ordre de développement des fils d'un noeud est primordial pour obtenir de bonnes performances !

Résultats des performances

Encadrement des performances de $\alpha\beta$

Le nombre de feuilles évaluées par $\alpha\beta$ est compris entre $l^{p/2}$ et l^p pour un arbre de largeur l et de profondeur p

Nombre de noeuds visités

$\alpha\beta$ visite au minimum un nombre de noeuds de l'ordre de $l^{p/2}$

Comparaison MiniMax / $\alpha\beta$

Au pire, $\alpha\beta$ explore tous les noeuds de l'arbre. Au mieux, $\alpha\beta$ peut voir à un horizon deux fois plus lointain que MiniMax dans le même temps.

L'ordre de développement des fils d'un noeud est primordial pour obtenir de bonnes performances !

Résultats des performances

Encadrement des performances de $\alpha\beta$

Le nombre de feuilles évaluées par $\alpha\beta$ est compris entre $l^{p/2}$ et l^p pour un arbre de largeur l et de profondeur p

Nombre de noeuds visités

$\alpha\beta$ visite au minimum un nombre de noeuds de l'ordre de $l^{p/2}$

Comparaison MiniMax / $\alpha\beta$

Au pire, $\alpha\beta$ explore tous les noeuds de l'arbre. Au mieux, $\alpha\beta$ peut voir à un horizon deux fois plus lointain que MiniMax dans le même temps.

L'ordre de développement des fils d'un noeud est primordial pour obtenir de bonnes performances !

Résultats des performances

Encadrement des performances de $\alpha\beta$

Le nombre de feuilles évaluées par $\alpha\beta$ est compris entre $l^{p/2}$ et l^p pour un arbre de largeur l et de profondeur p

Nombre de noeuds visités

$\alpha\beta$ visite au minimum un nombre de noeuds de l'ordre de $l^{p/2}$

Comparaison MiniMax / $\alpha\beta$

Au pire, $\alpha\beta$ explore tous les noeuds de l'arbre. Au mieux, $\alpha\beta$ peut voir à un horizon deux fois plus lointain que MiniMax dans le même temps.

L'ordre de développement des fils d'un noeud est primordial pour obtenir de bonnes performances !

À la recherche de performances

Problématique du temps réel

Questions liées au choix de la profondeur de recherche

La profondeur de recherche doit être fixée avant **de lancer la recherche** (L'horizon doit être équilibré sur tout l'arbre de recherche). Comment choisir la profondeur ?

- Comment garantir au joueur que l'ordinateur ne va pas passer trop de temps à réflechir ?
- Comment le garantir avec des machines différentes ?
- Comment lier le temps passé à réflechir avec la profondeur maximale de l'horizon ?
- Que se passe-t-il si l'arbre n'est pas homogène (hypothèse (très) réaliste) ?

Comment jouer au mieux dans un temps donné ?

Problématique du temps réel

Questions liées au choix de la profondeur de recherche

La profondeur de recherche doit être fixée avant de lancer la recherche (L'horizon doit être équilibré sur tout l'arbre de recherche). Comment choisir la profondeur ?

- Comment garantir au joueur que l'ordinateur ne va pas passer trop de temps à réflechir ?
- Comment le garantir avec des machines différentes ?
- Comment lier le temps passé à réflechir avec la profondeur maximale de l'horizon ?
- Que se passe-t-il si l'arbre n'est pas homogène (hypothèse (très) réaliste) ?

Comment jouer au mieux dans un temps donné ?

Problématique du temps réel

Questions liées au choix de la profondeur de recherche

La profondeur de recherche doit être fixée avant de lancer la recherche (L'horizon doit être équilibré sur tout l'arbre de recherche). Comment choisir la profondeur ?

- Comment garantir au joueur que l'ordinateur ne va pas passer trop de temps à réflechir ?
- Comment le garantir avec des machines différentes ?
- Comment lier le temps passé à réflechir avec la profondeur maximale de l'horizon ?
- Que se passe-t-il si l'arbre n'est pas homogène (hypothèse (très) réaliste) ?

Comment jouer au mieux dans un temps donné ?

Problématique du temps réel

Questions liées au choix de la profondeur de recherche

La profondeur de recherche doit être fixée avant de lancer la recherche (L'horizon doit être équilibré sur tout l'arbre de recherche). Comment choisir la profondeur ?

- Comment garantir au joueur que l'ordinateur ne va pas passer trop de temps à réflechir ?
- Comment le garantir avec des machines différentes ?
- Comment lier le temps passé à réflechir avec la profondeur maximale de l'horizon ?
- Que se passe-t-il si l'arbre n'est pas homogène (hypothèse (très) réaliste) ?

Comment jouer au mieux dans un temps donné ?

Iterative Deepening

Maîtriser le temps

Idée : Étendre peu à peu l'horizon de $\alpha\beta$

Mais doit-on mémoriser tout l'arbre pour repartir des feuilles ?

Iterative Deepening (ID)

Soit n initialisé à un horizon *immédiat*.

- Faire une recherche à horizon n
- S'il reste du temps, tout oublier (sauf le coup à jouer) et incrémenter n.

Et les performances ?

ID semble refaire beaucoup de fois la même chose. Est-ce pour autant très inefficace ?

Iterative Deepening

Maîtriser le temps

Idée : Étendre peu à peu l'horizon de $\alpha\beta$

Mais doit-on mémoriser tout l'arbre pour repartir des feuilles ?

Iterative Deepening (ID)

Soit n initialisé à un horizon *immédiat*.

- Faire une recherche à horizon n
- S'il reste du temps, tout oublier (sauf le coup à jouer) et incrémenter n.

Et les performances ?

ID semble refaire beaucoup de fois la même chose. Est-ce pour autant très inefficace ?

Iterative Deepening

Maîtriser le temps

Idée : Étendre peu à peu l'horizon de $\alpha\beta$

Mais doit-on mémoriser tout l'arbre pour repartir des feuilles ?

Iterative Deepening (ID)

Soit n initialisé à un horizon *immédiat*.

- Faire une recherche à horizon n
- S'il reste du temps, tout oublier (sauf le coup à jouer) et incrémenter n.

Et les performances ?

ID semble refaire beaucoup de fois la même chose. Est-ce pour autant très inefficace ?

Iterative Deepening

Maîtriser le temps

Idée : Étendre peu à peu l'horizon de $\alpha\beta$

Mais doit-on mémoriser tout l'arbre pour repartir des feuilles ?

Iterative Deepening (ID)

Soit n initialisé à un horizon *immédiat*.

- Faire une recherche à horizon n
- S'il reste du temps, tout oublier (sauf le coup à jouer) et incrémenter n.

Et les performances ?

ID semble refaire beaucoup de fois la même chose. Est-ce pour autant très inefficace ?

Iterative Deepening

Un algorithme pas si *inefficace*

La croissance exponentielle des arbres montre intuitivement que la majorité des noeuds (donc de la difficulté) se situent sur les feuilles...
Développer le dernier niveau coûte le plus cher, d'autant plus si on a un grand facteur de branchement.

Qui veut des chiffres ?

Une recherche à profondeur d et facteur de branchement b coûte

$$1 + b + b^2 + \ldots + b^{d-1} + b^d$$

Pour $(d = 5, b = 10)$ on a 11 111 noeuds. Si on fait un ID, on aurait développé 12 345 noeuds au total (1 pour d=1, 11 pour d=2, 111 pour d=3, 1 111 pour d=4 ...), soit 11% de noeuds supplémentaires (seulement).

Iterative Deepening

Un algorithme pas si *inefficace*

La croissance exponentielle des arbres montre intuitivement que la majorité des noeuds (donc de la difficulté) se situent sur les feuilles... Développer le dernier niveau coûte le plus cher, d'autant plus si on a un grand facteur de branchement.

Qui veut des chiffres ?

Une recherche à profondeur d et facteur de branchement b coûte

$$1 + b + b^2 + \ldots + b^{d-1} + b^d$$

Pour $(d = 5, b = 10)$ on a 11 111 noeuds. Si on fait un ID, on aurait développé 12 345 noeuds au total (1 pour d=1, 11 pour d=2, 111 pour d=3, 1 111 pour d=4 ...), soit 11% de noeuds supplémentaires (seulement).

Iterative Deepening

Un algorithme pas si *inefficace*

La croissance exponentielle des arbres montre intuitivement que la majorité des noeuds (donc de la difficulté) se situent sur les feuilles... Développer le dernier niveau coûte le plus cher, d'autant plus si on a un grand facteur de branchement.

Qui veut des chiffres ?

Une recherche à profondeur d et facteur de branchement b coûte

$$1 + b + b^2 + \ldots + b^{d-1} + b^d$$

Pour $(d = 5, b = 10)$ on a 11 111 noeuds. Si on fait un ID, on aurait développé 12 345 noeuds au total (1 pour d=1, 11 pour d=2, 111 pour d=3, 1 111 pour d=4 ...), soit 11% de noeuds supplémentaires (seulement).

Iterative Deepening

Un algorithme pas si *inefficace*

La croissance exponentielle des arbres montre intuitivement que la majorité des noeuds (donc de la difficulté) se situent sur les feuilles... Développer le dernier niveau coûte le plus cher, d'autant plus si on a un grand facteur de branchement.

Qui veut des chiffres ?

Une recherche à profondeur d et facteur de branchement b coûte

$$1 + b + b^2 + \ldots + b^{d-1} + b^d$$

Pour $(d = 5, b = 10)$ on a 11 111 noeuds. Si on fait un ID, on aurait développé 12 345 noeuds au total (1 pour d=1, 11 pour d=2, 111 pour d=3, 1 111 pour d=4 ...), soit 11% de noeuds supplémentaires (seulement).

Iterative Deepening

Des avantages de poids avec $\alpha\beta$

Avantages

- Grande souplesse dans la gestion du temps. Le programme peut donner la meilleur valeur trouvée à n'importe quel moment.
- Couplé à $\alpha\beta$, ID peut même devenir plus efficace qu'un seul $\alpha\beta$

Comment ?

$\alpha\beta$ élague d'autant plus que les meilleurs coups sont développés en premier. On profite donc de l'ancien $\alpha\beta$ à profondeur n pour ordonner les fils à profondeur $n+1$.

Iterative Deepening

Des avantages de poids avec $\alpha\beta$

Avantages

- Grande souplesse dans la gestion du temps. Le programme peut donner la meilleur valeur trouvée à n'importe quel moment.
- Couplé à $\alpha\beta$, ID peut même devenir plus efficace qu'un seul $\alpha\beta$

Comment ?

$\alpha\beta$ élague d'autant plus que les meilleurs coups sont développés en premier. On profite donc de l'ancien $\alpha\beta$ à profondeur n pour ordonner les fils à profondeur $n+1$.

Iterative Deepening

En pratique

Idées

Plutôt que de tout oublier après chaque relancement de ID, on va essayer de mémoriser pour réordonner l'arbre ensuite.

Première méthode

Garder tout l'arbre de recherche en mémoire, l'ordonner à chaque niveau puis relancer $\alpha\beta$ directement sur le nouvel arbre *optimal*.

Seconde méthode

Réserver une zone mémoire pour associer à chaque plateau de jeu déjà vu le meilleur des fils et la valeur heuristique associée (ainsi que sa profondeur).

Iterative Deepening

En pratique

Idées

Plutôt que de tout oublier après chaque relancement de ID, on va essayer de mémoriser pour réordonner l'arbre ensuite.

Première méthode

Garder tout l'arbre de recherche en mémoire, l'ordonner à chaque niveau puis relancer $\alpha\beta$ directement sur le nouvel arbre *optimal*.

Seconde méthode

Réserver une zone mémoire pour associer à chaque plateau de jeu déjà vu le meilleur des fils et la valeur heuristique associée (ainsi que sa profondeur).

Iterative Deepening

En pratique

Idées

Plutôt que de tout oublier après chaque relancement de ID, on va essayer de mémoriser pour réordonner l'arbre ensuite.

Première méthode

Garder tout l'arbre de recherche en mémoire, l'ordonner à chaque niveau puis relancer $\alpha\beta$ directement sur le nouvel arbre *optimal*.

Seconde méthode

Réserver une zone mémoire pour associer à chaque plateau de jeu déjà vu le meilleur des fils et la valeur heuristique associée (ainsi que sa profondeur).

Iterative Deepening

En pratique

Idées

Plutôt que de tout oublier après chaque relancement de ID, on va essayer de mémoriser pour réordonner l'arbre ensuite.

Première méthode

Garder tout l'arbre de recherche en mémoire, l'ordonner à chaque niveau puis relancer $\alpha\beta$ directement sur le nouvel arbre *optimal*.

Seconde méthode

Réserver une zone mémoire pour associer à chaque plateau de jeu déjà vu le meilleur des fils et la valeur heuristique associée (ainsi que sa profondeur).

Repérer des coups profonds

Question

Comment repérer efficacement d'éventuelles stratégies gagnantes au delà de l'horizon ?

Idée

Il faut élaguer plus encore dans $\alpha\beta$. On réduit la valeur heuristique à $[0, 1]$ pour couper plus et aller plus loin. Découper le temps T que l'on a pour un coup en 3 :

- Chercher un coup gagnant profond par un $\alpha\beta$ sur $[0, 1]$. Y passer $T/10$ secondes
- Si on ne trouve pas de coups gagnant, trouver un *bon coup* avec un $\alpha\beta$ classique en $8T/10$ secondes.
- Vérifier en $T/10$ secondes que ce coup n'est pas un coup perdant à un horizon lointain

Repérer des coups profonds

Question

Comment repérer efficacement d'éventuelles stratégies gagnantes au delà de l'horizon ?

Idée

Il faut élaguer plus encore dans $\alpha\beta$. On réduit la valeur heuristique à $[0, 1]$ pour couper plus et aller plus loin. Découper le temps T que l'on a pour un coup en 3 :

- Chercher un coup gagnant profond par un $\alpha\beta$ sur $[0, 1]$. Y passer $T/10$ secondes
- Si on ne trouve pas de coups gagnant, trouver un *bon coup* avec un $\alpha\beta$ classique en $8T/10$ secondes.
- Vérifier en $T/10$ secondes que ce coup n'est pas un coup perdant à un horizon lointain

Repérer des coups profonds

Question

Comment repérer efficacement d'éventuelles stratégies gagnantes au delà de l'horizon ?

Idée

Il faut élaguer plus encore dans $\alpha\beta$. On réduit la valeur heuristique à $[0, 1]$ pour couper plus et aller plus loin. Découper le temps T que l'on a pour un coup en 3 :

- Chercher un coup gagnant profond par un $\alpha\beta$ sur $[0, 1]$. Y passer $T/10$ secondes
- Si on ne trouve pas de coups gagnant, trouver un *bon coup* avec un $\alpha\beta$ classique en $8T/10$ secondes.
- Vérifier en $T/10$ secondes que ce coup n'est pas un coup perdant à un horizon lointain

Repérer des coups profonds

Question

Comment repérer efficacement d'éventuelles stratégies gagnantes au delà de l'horizon ?

Idée

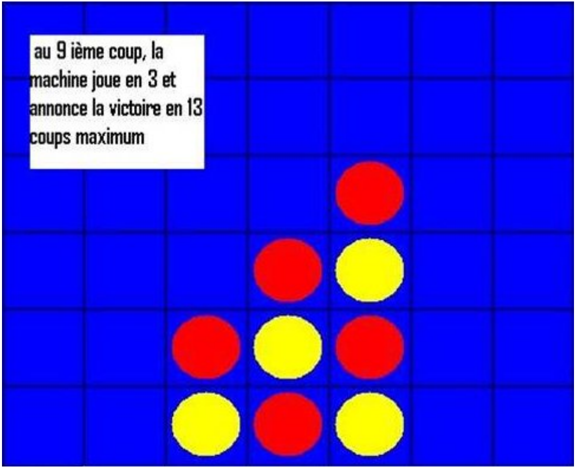
Il faut élaguer plus encore dans $\alpha\beta$. On réduit la valeur heuristique à $[0, 1]$ pour couper plus et aller plus loin. Découper le temps T que l'on a pour un coup en 3 :

- Chercher un coup gagnant profond par un $\alpha\beta$ sur $[0, 1]$. Y passer $T/10$ secondes
- Si on ne trouve pas de coups gagnant, trouver un *bon coup* avec un $\alpha\beta$ classique en $8T/10$ secondes.
- Vérifier en $T/10$ secondes que ce coup n'est pas un coup perdant à un horizon lointain

Repérer les coups fatals

Exemple sur le Puissance 4

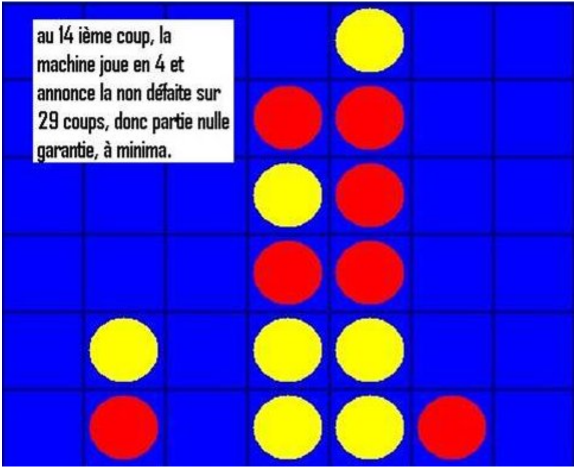
Images issues d'une applet en Java disponible sur le web.



Repérer les coups fatals

Exemple sur le Puissance 4

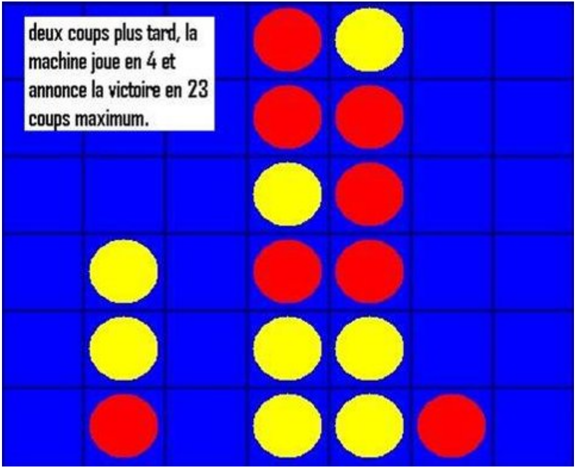
Images issues d'une applet en Java disponible sur le web.



Repérer les coups fatals

Exemple sur le Puissance 4

Images issues d'une applet en Java disponible sur le web.



Problèmes à l'horizon

Inconvénients de fixer un horizon

Tous les programmes doivent fixer un horizon. Pourtant, cela entraîne deux problèmes importants :

- Manque de clairvoyance : On ne peut prévoir un coup (même fatal) au-delà de l'horizon
- Aveuglement volontaire : l'approche MiniMax va pousser le programme à tout faire pour repousser un coup mauvais – mais inévitable – au delà de l'horizon.

Deux solutions... Parmi d'autres

- Déséquilibrer l'arbre de recherche
- Utiliser une *méta-heuristique*

Problèmes à l'horizon

Inconvénients de fixer un horizon

Tous les programmes doivent fixer un horizon. Pourtant, cela entraîne deux problèmes importants :

- Manque de clairvoyance : On ne peut prévoir un coup (même fatal) au-delà de l'horizon
- Aveuglement volontaire : l'approche MiniMax va pousser le programme à tout faire pour repousser un coup mauvais – mais inévitable – au delà de l'horizon.

Deux solutions... Parmi d'autres

- Déséquilibrer l'arbre de recherche
- Utiliser une *méta-heuristique*

Problèmes à l'horizon

Inconvénients de fixer un horizon

Tous les programmes doivent fixer un horizon. Pourtant, cela entraîne deux problèmes importants :

- Manque de clairvoyance : On ne peut prévoir un coup (même fatal) au-delà de l'horizon
- Aveuglement volontaire : l'approche MiniMax va pousser le programme à tout faire pour repousser un coup mauvais – mais inévitable – au delà de l'horizon.

Deux solutions... Parmi d'autres

- Déséquilibrer l'arbre de recherche
- Utiliser une *méta-heuristique*

Problèmes à l'horizon

Inconvénients de fixer un horizon

Tous les programmes doivent fixer un horizon. Pourtant, cela entraîne deux problèmes importants :

- Manque de clairvoyance : On ne peut prévoir un coup (même fatal) au-delà de l'horizon
- Aveuglement volontaire : l'approche MiniMax va pousser le programme à tout faire pour repousser un coup mauvais – mais inévitable – au delà de l'horizon.

Deux solutions... Parmi d'autres

- Déséquilibrer l'arbre de recherche
- Utiliser une *méta-heuristique*

Atténuation d'horizons

Zoomer où l'on va aller

Idée

Une fois que l'on a fait le choix du coup à jouer, passer un peu de temps pour voir si la fonction heuristique ne nous a pas trompé. On va déséquilibrer l'arbre de recherche.

Deux solutions... Parmi d'autres

- Redévelopper un $\alpha\beta$ sur la branche sélectionnée.
- Redévelopper un $\alpha\beta$ sur la meilleure feuille visée.
- Redévelopper un $\alpha\beta$ sur un sous-arbre *intéressant*

Atténuation d'horizons

Zoomer où l'on va aller

Idée

Une fois que l'on a fait le choix du coup à jouer, passer un peu de temps pour voir si la fonction heuristique ne nous a pas trompé. On va déséquilibrer l'arbre de recherche.

Deux solutions... Parmi d'autres

- Redévelopper un $\alpha\beta$ sur la branche sélectionnée.
- Redévelopper un $\alpha\beta$ sur la meilleure feuille visée.
- Redévelopper un $\alpha\beta$ sur un sous-arbre *intéressant*

Atténuation d'horizons

Zoomer où l'on va aller

Idée

Une fois que l'on a fait le choix du coup à jouer, passer un peu de temps pour voir si la fonction heuristique ne nous a pas trompé. On va déséquilibrer l'arbre de recherche.

Deux solutions... Parmi d'autres

- Redévelopper un $\alpha\beta$ sur la branche sélectionnée.
- Redévelopper un $\alpha\beta$ sur la meilleure feuille visée.
- Redévelopper un $\alpha\beta$ sur un sous-arbre *intéressant*

Atténuation d’horizons

Zoomer où l’on va aller

Idée

Une fois que l’on a fait le choix du coup à jouer, passer un peu de temps pour voir si la fonction heuristique ne nous a pas trompé. On va déséquilibrer l’arbre de recherche.

Deux solutions... Parmi d’autres

- Redévelopper un $\alpha\beta$ sur la branche sélectionnée.
- Redévelopper un $\alpha\beta$ sur la meilleure feuille visée.
- Redévelopper un $\alpha\beta$ sur un sous-arbre *intéressant*

Atténuation d'horizons

Reconsidérer la notion d'horizons

Idée

Définir l'horizon en fonction de l'intérêt des coups joués et non en fonction du nombre de coups.

Principes

On part par exemple avec une profondeur $SX = 10.p$, où p est la profondeur au sens habituel.

- Un coup « moyen » diminue SX de 10.
- Un coup intéressant (prise de pièce, échec) ne diminue SX que de 2 ou 3.
- Un coup peu intéressant diminue beaucoup SX (de l'ordre de 35).

Résultat : les coups intéressants seront explorés plus profondément.

Atténuation d'horizons

Reconsidérer la notion d'horizons

Idée

Définir l'horizon en fonction de l'intérêt des coups joués et non en fonction du nombre de coups.

Principes

On part par exemple avec une profondeur $SX = 10.p$, où p est la profondeur au sens habituel.

- Un coup « moyen » diminue SX de 10.
- Un coup intéressant (prise de pièce, échec) ne diminue SX que de 2 ou 3.
- Un coup peu intéressant diminue beaucoup SX (de l'ordre de 35).

Résultat : les coups intéressants seront explorés plus profondément.

Analyse des parties

Passer du temps là où il faut

Concentrer l'effort de recherche dans les *zones critiques*

- Si tous les débuts de partie se ressemblent, ne pas y consacrer trop de temps (ex : jouer au hasard pour le jeu des *amazones*)
- Passer plus de temps au milieu de partie (ex Awalé) car les fins de partie ne demandent pas beaucoup de temps (élagage très efficace dans l'Awalé).

Mais seule une connaissance profonde du jeu permet d'affiner les moments où la recherche de bon coups est la plus critique.

Analyse des parties

Passer du temps là où il faut

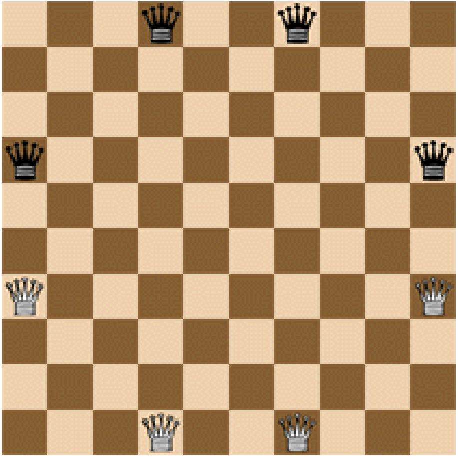
Concentrer l'effort de recherche dans les *zones critiques*

- Si tous les débuts de partie se ressemblent, ne pas y consacrer trop de temps (ex : jouer au hasard pour le jeu des *amazones*)
- Passer plus de temps au milieu de partie (ex Awalé) car les fins de partie ne demandent pas beaucoup de temps (élagage très efficace dans l'Awalé).

Mais seule une connaissance profonde du jeu permet d'affiner les moments où la recherche de bon coups est la plus critique.

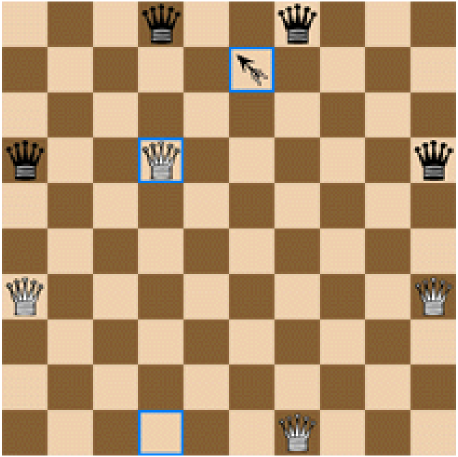
Exemple sur les Amazones

Comment gérer l'ouverture ?



Exemple sur les Amazones

Comment gérer l'ouverture ?



Analyse des parties

Passer du temps là où il faut

3 phases communes à tous les jeux

- Début de partie
- Milieu de partie
- Fin de partie

Un bon programme de jeu doit avoir un comportement fondamentalement différent dans chacune de ces trois phases.

Encore une heuristique

Il faut donc avoir une heuristique de haut niveau permettant de choisir entre les différentes heuristiques liées aux phases du jeu.

Analyse des parties

Passer du temps là où il faut

3 phases communes à tous les jeux

- Début de partie
- Milieu de partie
- Fin de partie

Un bon programme de jeu doit avoir un comportement fondamentalement différent dans chacune de ces trois phases.

Encore une heuristique

Il faut donc avoir une heuristique de haut niveau permettant de choisir entre les différentes heuristiques liées aux phases du jeu.

Agir sur les heuristiques

On peut faire évoluer la valeur de la fonction heuristique suivant l'avancement dans le jeu.

Exemples

- Awalé : faire des greniers en début de partie, puis simplement compter les graines en fin de partie
- Othello : Prendre des cases stratégiques, puis compter les pions.
- ...

La transition entre les différentes fonctions heuristiques doit être douce pour que l'horizon évolue doucement.

Début de partie : bibliothèques d'ouverture

Pour concevoir une bibliothèque, deux solutions :

- Utilisation d'un expert (recopier des ouvertures dans la littérature).
- Construction de ses propres bibliothèques. Utile lorsque le jeu n'a pas déjà été grandement étudié.
  - Lancement d'$\alpha\beta$ à grande profondeur (extension hors-ligne de l'horizon des premiers coups)
    Faire jouer différentes ouvertures contre le programme lui-même

Comment stocker la bibliothèque ?

- Soit sous forme d'automate
- Soit sous forme de tables de transposition

Ne pas oublier d'introduire du hasard...

Début de partie : bibliothèques d'ouverture

Pour concevoir une bibliothèque, deux solutions :

- Utilisation d'un expert (recopier des ouvertures dans la littérature).
- Construction de ses propres bibliothèques. Utile lorsque le jeu n'a pas déjà été grandement étudié.
  - Lancement d'$\alpha\beta$ à grande profondeur (extension hors-ligne de l'horizon des premiers coups)
  - Faire jouer différentes ouvertures contre le programme lui-même

Comment stocker la bibliothèque ?

- Soit sous forme d'automate
- Soit sous forme de tables de transposition

Ne pas oublier d'introduire du hasard...

Début de partie : bibliothèques d'ouverture

Pour concevoir une bibliothèque, deux solutions :

- Utilisation d'un expert (recopier des ouvertures dans la littérature).
- Construction de ses propres bibliothèques. Utile lorsque le jeu n'a pas déjà été grandement étudié.
  - Lancement d'$\alpha\beta$ à grande profondeur (extension hors-ligne de l'horizon des premiers coups)
  - Faire jouer différentes ouvertures contre le programme lui-même

Comment stocker la bibliothèque ?

- Soit sous forme d'automate
- Soit sous forme de tables de transposition

Ne pas oublier d'introduire du hasard...

Début de partie : bibliothèques d'ouverture

Pour concevoir une bibliothèque, deux solutions :

- Utilisation d'un expert (recopier des ouvertures dans la littérature).
- Construction de ses propres bibliothèques. Utile lorsque le jeu n'a pas déjà été grandement étudié.
  - Lancement d'$\alpha\beta$ à grande profondeur (extension hors-ligne de l'horizon des premiers coups)
  - Faire jouer différentes ouvertures contre le programme lui-même

Comment stocker la bibliothèque ?

- Soit sous forme d'automate
- Soit sous forme de tables de transposition

Ne pas oublier d'introduire du hasard...

Début de partie : bibliothèques d'ouverture

Pour concevoir une bibliothèque, deux solutions :

- Utilisation d'un expert (recopier des ouvertures dans la littérature).
- Construction de ses propres bibliothèques. Utile lorsque le jeu n'a pas déjà été grandement étudié.
  - Lancement d'$\alpha\beta$ à grande profondeur (extension hors-ligne de l'horizon des premiers coups)
  - Faire jouer différentes ouvertures contre le programme lui-même

Comment stocker la bibliothèque ?

- Soit sous forme d'automate
- Soit sous forme de tables de transposition

Ne pas oublier d'introduire du hasard...

Milieu de partie

Domaine de $\alpha\beta$ avec tous les mécanismes vus :

- Recherche de coups profond
- Iterative Deepening
- Horizon de coups intéressants
- ...

Milieu de partie

Domaine de $\alpha\beta$ avec tous les mécanismes vus :

- Recherche de coups profond
- Iterative Deepening
- Horizon de coups intéressants
- ...

Fin de partie

Le domaine de la puissance brute

Généralement, le jeu est très contraint. Le facteur de branchement de l'arbre de jeu est réduit. On essaye d'aller jusqu'au bout de l'arbre de recherche pour trouver une stratégie gagnante.

- Les heuristiques sont généralement réduites à « gagné » « perdu ».
- Il faut penser à augmenter considérablement l'horizon.

Question

Peut-on décider si on a gagné ou perdu sans aller au bout de l'arbre ?

Fin de partie

Le domaine de la puissance brute

Généralement, le jeu est très contraint. Le facteur de branchement de l'arbre de jeu est réduit. On essaye d'aller jusqu'au bout de l'arbre de recherche pour trouver une stratégie gagnante.

- Les heuristiques sont généralement réduites à « gagné » « perdu ».
- Il faut penser à augmenter considérablement l'horizon.

Question

Peut-on décider si on a gagné ou perdu sans aller au bout de l'arbre ?

Fin de partie

Le domaine de la puissance brute

Généralement, le jeu est très contraint. Le facteur de branchement de l'arbre de jeu est réduit. On essaye d'aller jusqu'au bout de l'arbre de recherche pour trouver une stratégie gagnante.

- Les heuristiques sont généralement réduites à « gagné » « perdu ».
- Il faut penser à augmenter considérablement l'horizon.

Question

Peut-on décider si on a gagné ou perdu sans aller au bout de l'arbre ?

Fin de partie

Le domaine de la puissance brute

Généralement, le jeu est très contraint. Le facteur de branchement de l'arbre de jeu est réduit. On essaye d'aller jusqu'au bout de l'arbre de recherche pour trouver une stratégie gagnante.

- Les heuristiques sont généralement réduites à « gagné » « perdu ».
- Il faut penser à augmenter considérablement l'horizon.

Question

Peut-on décider si on a gagné ou perdu sans aller au bout de l'arbre ?

Fin de partie

Le domaine de la puissance brute

Généralement, le jeu est très contraint. Le facteur de branchement de l'arbre de jeu est réduit. On essaye d'aller jusqu'au bout de l'arbre de recherche pour trouver une stratégie gagnante.

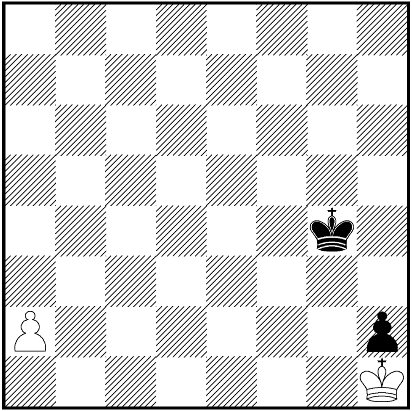
- Les heuristiques sont généralement réduites à « gagné » « perdu ».
- Il faut penser à augmenter considérablement l'horizon.

Question

Peut-on décider si on a gagné ou perdu sans aller au bout de l'arbre ?

Fin de partie

On n'en est pas arrivé là pour perdre (1/3) !

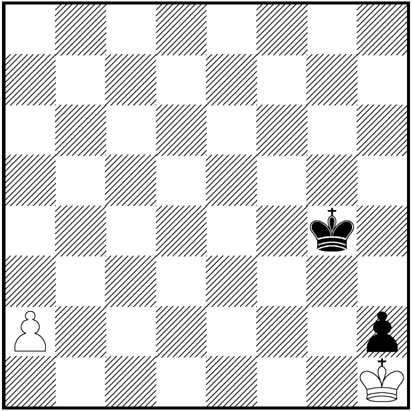


Blanc joue.

Un problème d'horizon trompe Noir : il faudrait voir à 10 demi-coups.

Fin de partie

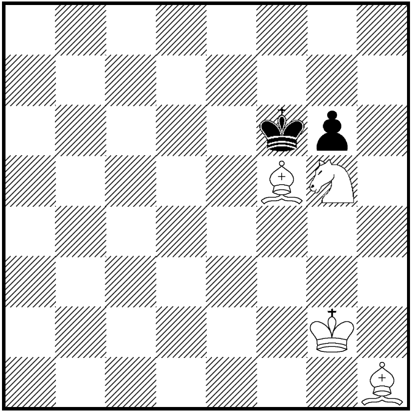
On n'en est pas arrivé là pour perdre (1/3) !



Blanc joue.
Un problème d'horizon trompe Noir : il faudrait voir à 10 demi-coups.

Fin de partie

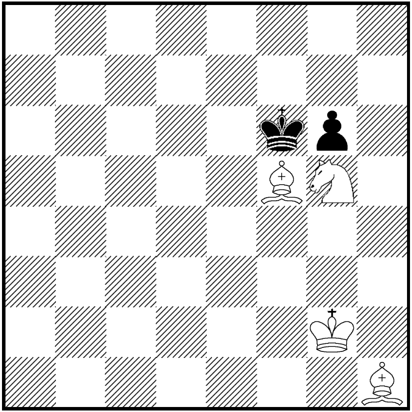
On n'en est pas arrivé là pour perdre (2/3) !



Blanc joue et doit perdre son fou ou son cheval. Il sacrifie son fou pour prendre le pion. La Finale est Roi, Fou, Cavalier contre Roi, Gagnante

Fin de partie

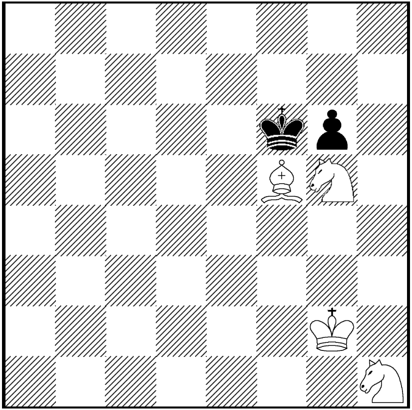
On n'en est pas arrivé là pour perdre (2/3) !



Blanc joue et doit perdre son fou ou son cheval. Il sacrifie son fou pour prendre le pion. La Finale est Roi, Fou, Cavalier contre Roi, Gagnante

Fin de partie

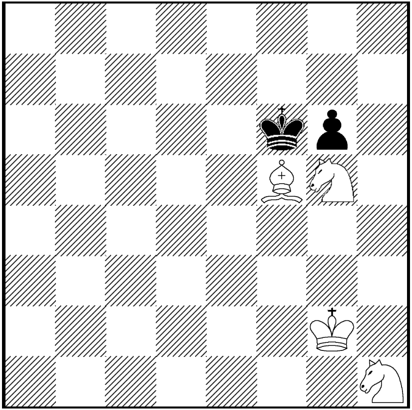
On n'en est pas arrivé là pour perdre (3/3) !



Blanc joue et va faire comme précédemment. Il sacrifie son fou pour prendre le pion. La Finale est Roi, Cavalier, Cavalier contre Roi, Nulle Il fallait mettre son Fou en d3 !

Fin de partie

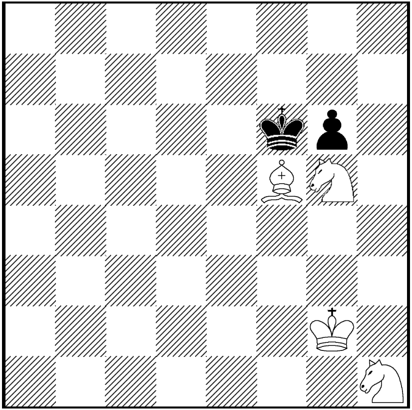
On n'en est pas arrivé là pour perdre (3/3) !



Blanc joue et va faire comme précédemment. Il sacrifie son fou pour prendre le pion. La Finale est Roi, Cavalier, Cavalier contre Roi, Nulle Il fallait mettre son Fou en d3 !

Fin de partie

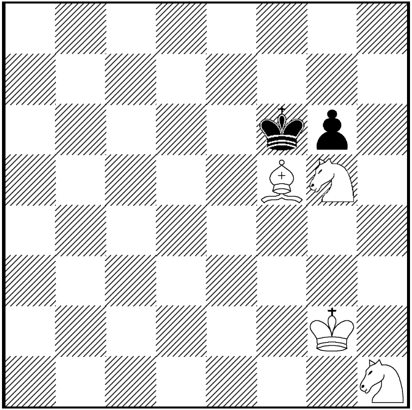
On n'en est pas arrivé là pour perdre (3/3) !



Blanc joue et va faire comme précédemment. Il sacrifie son fou pour prendre le pion. La Finale est Roi, Cavalier, Cavalier contre Roi, Nulle Il fallait mettre son Fou en d3 !

Fin de partie

On n'en est pas arrivé là pour perdre (3/3) !



Blanc joue et va faire comme précédemment. Il sacrifie son fou pour prendre le pion. La Finale est Roi, Cavalier, Cavalier contre Roi, Nulle Il fallait mettre son Fou en d3 !

Fin de partie

D'autres cas pathologiques aux échecs : certaines finales imposent un plan très précis pour faire Mat en moins de 50 coups.
Il faut savoir reconnaître des fins de parties gagnantes ou perdantes.
Il faut construire une bibliothèque de fermeture soit :

- par un expert
- par analyse rétrograde de toutes les positions finales du jeu

Question

Comment vérifier efficacement si un plateau de jeu a déjà été vu ?

Fin de partie

D'autres cas pathologiques aux échecs : certaines finales imposent un plan très précis pour faire Mat en moins de 50 coups.
Il faut savoir reconnaître des fins de parties gagnantes ou perdantes.
Il faut construire une bibliothèque de fermeture soit :

- par un expert
- par analyse rétrograde de toutes les positions finales du jeu

Question

Comment vérifier efficacement si un plateau de jeu a déjà été vu ?

Heuristiques, avancées

Calculer ou maintenir l'heuristique ?

Question

Faut-il :

- calculer la fonction heuristique entièrement à chaque feuille ?
  - pas de calcul heuristique aux noeuds internes de l'arbre
  - faire tout le calcul à chaque feuille
- maintenir la valeur heuristique d'un plateau après chaque coups ?
  - calcul heuristique à chaque mouvement dans l'arbre
  - aucun calcul aux feuilles de l'arbre

Exemple ?

Calculer ou maintenir, des éléments de réponse

Soient

- T_f le temps passé à calculer entièrement la fonction heuristique
- T_m le temps passé à mettre à jour la fonction heuristique après un coup
- b le facteur de branchement de l'arbre et p sa profondeur

(*rappel*) : L'arbre contient b^p noeuds et b^{p-1} feuilles. Il faut comparer $T_{sol_1} = T_f.b^{p-1}$ et $T_{sol_2} = T_m.b^p$. Mettre à jour est plus intéressant si

$$T_m.b^p < T_f.b^{p-1}$$

c-a-d si

$$T_m < \frac{T_f}{b}$$

ce qui peut être facilement obtenu sur certains jeux

Calculer ou maintenir, des éléments de réponse

Soient

- T_f le temps passé à calculer entièrement la fonction heuristique
- T_m le temps passé à mettre à jour la fonction heuristique après un coup
- b le facteur de branchement de l'arbre et p sa profondeur

(*rappel*) : L'arbre contient b^p noeuds et b^{p-1} feuilles. Il faut comparer $T_{sol_1} = T_f.b^{p-1}$ et $T_{sol_2} = T_m.b^p$. Mettre à jour est plus intéressant si

$$T_m.b^p < T_f.b^{p-1}$$

c-a-d si

$$T_m < \frac{T_f}{b}$$

ce qui peut être facilement obtenu sur certains jeux

Calculer ou maintenir, des éléments de réponse

Soient

- T_f le temps passé à calculer entièrement la fonction heuristique
- T_m le temps passé à mettre à jour la fonction heuristique après un coup
- b le facteur de branchement de l'arbre et p sa profondeur

(*rappel*) : L'arbre contient b^p noeuds et b^{p-1} feuilles. Il faut comparer $T_{sol_1} = T_f.b^{p-1}$ et $T_{sol_2} = T_m.b^p$. Mettre à jour est plus intéressant si

$$T_m.b^p < T_f.b^{p-1}$$

c-a-d si

$$T_m < \frac{T_f}{b}$$

ce qui peut être facilement obtenu sur certains jeux

Calculer ou maintenir, des éléments de réponse

Soient

- T_f le temps passé à calculer entièrement la fonction heuristique
- T_m le temps passé à mettre à jour la fonction heuristique après un coup
- b le facteur de branchement de l'arbre et p sa profondeur

(*rappel*) : L'arbre contient b^p noeuds et b^{p-1} feuilles. Il faut comparer $T_{sol_1} = T_f.b^{p-1}$ et $T_{sol_2} = T_m.b^p$. Mettre à jour est plus intéressant si

$$T_m.b^p < T_f.b^{p-1}$$

c-a-d si

$$T_m < \frac{T_f}{b}$$

ce qui peut être facilement obtenu sur certains jeux

Coups spéciaux : *Coups meurtriers*

Idée

Si la valeur heuristique d'un coup chute brutalement, c'est que le coup joué est *meurtrier*.

En pratique : Une fois les coups meurtriers identifiés, les développer en priorité dans $\alpha\beta$.

Problèmes

Il faut arriver à repérer d'un coup sur l'autre les coups dont la valeur heuristique a brutalement chuté. Il faut utiliser des tables de transpostions (comme dans *Iterative Deepening*).

Coups spéciaux : *Coups meurtriers*

Idée

Si la valeur heuristique d'un coup chute brutalement, c'est que le coup joué est *meurtrier*.

En pratique : Une fois les coups meurtriers identifiés, les développer en priorité dans $\alpha\beta$.

Problèmes

Il faut arriver à repérer d'un coup sur l'autre les coups dont la valeur heuristique a brutalement chuté. Il faut utiliser des tables de transpostions (comme dans *Iterative Deepening*).

Coups spéciaux : *Coups meurtriers*

Idée

Si la valeur heuristique d'un coup chute brutalement, c'est que le coup joué est *meurtrier*.

En pratique : Une fois les coups meurtriers identifiés, les développer en priorité dans $\alpha\beta$.

Problèmes

Il faut arriver à repérer d'un coup sur l'autre les coups dont la valeur heuristique a brutalement chuté. Il faut utiliser des tables de transpostions (comme dans *Iterative Deepening*).

Coups spéciaux : *Coups meurtriers*

Idée

Si la valeur heuristique d'un coup chute brutalement, c'est que le coup joué est *meurtrier*.

En pratique : Une fois les coups meurtriers identifiés, les développer en priorité dans $\alpha\beta$.

Problèmes

Il faut arriver à repérer d'un coup sur l'autre les coups dont la valeur heuristique a brutalement chuté. Il faut utiliser des tables de transpostions (comme dans *Iterative Deepening*).

Utilisation d'une *méta-heuristique*

Repérer les *zones calmes*

Idée

Si on évalue une position alors que des pièces peuvent encore être prises, ou sont en danger immédiat, on n'a pas de bonne estimation.

Problèmes

Il faut arriver à estimer la pertinence de la fonction heuristique que l'on avait. Cette mesure de pertinence est elle-même une heuristique. D'où le nom de *méta-heuristique*.

En pratique : Tant que des pièces peuvent être prises, la fonction heuristique va dérouler explicitement les prises les plus importantes jusqu'au bout avant évaluation par la fonction heuristique habituelle. C'est le seul cas où une fonction heuristique simule des coups pour renvoyer un résultat.

Utilisation d'une *méta-heuristique*

Repérer les *zones calmes*

Idée

Si on évalue une position alors que des pièces peuvent encore être prises, ou sont en danger immédiat, on n'a pas de bonne estimation.

Problèmes

Il faut arriver à estimer la pertinence de la fonction heuristique que l'on avait. Cette mesure de pertinence est elle-même une heuristique. D'où le nom de *méta-heuristique*.

En pratique : Tant que des pièces peuvent être prises, la fonction heuristique va dérouler explicitement les prises les plus importantes jusqu'au bout avant évaluation par la fonction heuristique habituelle. C'est le seul cas où une fonction heuristique simule des coups pour renvoyer un résultat.

Utilisation d'une *méta-heuristique*

Repérer les *zones calmes*

Idée

Si on évalue une position alors que des pièces peuvent encore être prises, ou sont en danger immédiat, on n'a pas de bonne estimation.

Problèmes

Il faut arriver à estimer la pertinence de la fonction heuristique que l'on avait. Cette mesure de pertinence est elle-même une heuristique. D'où le nom de *méta-heuristique*.

En pratique : Tant que des pièces peuvent être prises, la fonction heuristique va dérouler explicitement les prises les plus importantes jusqu'au bout avant évaluation par la fonction heuristique habituelle. C'est le seul cas où une fonction heuristique simule des coups pour renvoyer un résultat.

Utilisation d'une *méta-heuristique*

Repérer les *zones calmes*

Idée

Si on évalue une position alors que des pièces peuvent encore être prises, ou sont en danger immédiat, on n'a pas de bonne estimation.

Problèmes

Il faut arriver à estimer la pertinence de la fonction heuristique que l'on avait. Cette mesure de pertinence est elle-même une heuristique. D'où le nom de *méta-heuristique*.

En pratique : Tant que des pièces peuvent être prises, la fonction heuristique va dérouler explicitement les prises les plus importantes jusqu'au bout avant évaluation par la fonction heuristique habituelle.
C'est le seul cas où une fonction heuristique simule des coups pour renvoyer un résultat.

Utilisation d'une *méta-heuristique*

Repérer les *zones calmes*

Idée

Si on évalue une position alors que des pièces peuvent encore être prises, ou sont en danger immédiat, on n'a pas de bonne estimation.

Problèmes

Il faut arriver à estimer la pertinence de la fonction heuristique que l'on avait. Cette mesure de pertinence est elle-même une heuristique. D'où le nom de *méta-heuristique*.

En pratique : Tant que des pièces peuvent être prises, la fonction heuristique va dérouler explicitement les prises les plus importantes jusqu'au bout avant évaluation par la fonction heuristique habituelle. C'est le seul cas où une fonction heuristique simule des coups pour renvoyer un résultat.

Coups trivialement inintéressants

Heuristique de coup nul

Idée

Si un joueur peut joueur deux coups de suite et que la valeur heuristique du damier ne change pas beaucoup, le premier coup peut être écarté.

Problèmes :Dans certains cas, des parties (trop) importantes de l'arbre de recherche sont élaguées.

Coups trivialement inintéressants

Heuristique de coup nul

Idée

Si un joueur peut joueur deux coups de suite et que la valeur heuristique du damier ne change pas beaucoup, le premier coup peut être écarté.

Problèmes :Dans certains cas, des parties (trop) importantes de l'arbre de recherche sont élaguées.

Apprendre avant de jouer

Idée

Commencer avec une heuristique idiote et laisser le programme apprendre par lui-même une bonne fonction heuristique, en jouant contre lui-même.

Apprendre pendant le jeu

Idée

La différence entre l'estimation d'un noeud et sa vraie valeur (que l'on découvre plus tard) sert de base à l'apprentissage.

Recherches sur fenêtres réduites

Résultats sur $\alpha\beta$

Définition

On appelle fenêtre $\alpha\beta$ le couple $[\alpha, \beta]$ où α et β sont les deux paramètres de la procédure $\alpha\beta$.

Propriété de la taille des fenêtres

Soient deux fenêtres $[\alpha_1, \beta_1]$ et $[\alpha_2, \beta_2]$ telles que $\alpha_1 \leq \alpha_2 < \beta_2 \leq \beta_1$, l'algorithme $\alpha\beta$ appelé avec $[\alpha_2, \beta_2]$ explore moins de noeuds que celui appelé avec la fenêtre $[\alpha_1, \beta_1]$.

Résultats sur $\alpha\beta$

Définition

On appelle fenêtre $\alpha\beta$ le couple $[\alpha, \beta]$ où α et β sont les deux paramètres de la procédure $\alpha\beta$.

Propriété de la taille des fenêtres

Soient deux fenêtres $[\alpha_1, \beta_1]$ et $[\alpha_2, \beta_2]$ telles que $\alpha_1 \leq \alpha_2 < \beta_2 \leq \beta_1$, l'algorithme $\alpha\beta$ appelé avec $[\alpha_2, \beta_2]$ explore moins de noeuds que celui appelé avec la fenêtre $[\alpha_1, \beta_1]$.

$\alpha\beta$ pour tester la valeur MiniMax

Optimalité de $\alpha\beta$

- $\alpha\beta$ est optimal à ordre de parcours fixé, à un polynôme prêt.
- $\alpha\beta$ appelé avec la fenêtre $[v, v+1]$ est optimal pour savoir si la valeur de la racine est supérieure ou égale à v.

$\alpha\beta$ avec une fenêtre d'appel $[v, v+1]$ peut être utilisé efficacement pour *tester* la valeur MiniMax d'un arbre de jeu.

```
Fonction Test(etat, v)                ▷ Test si MiniMax(Etat) > v
    val ← NegEchecαβ(s, v, v + 1)
    Retourner (val > v)
Fin Fonction
```

$\alpha\beta$ pour tester la valeur MiniMax

Optimalité de $\alpha\beta$

- $\alpha\beta$ est optimal à ordre de parcours fixé, à un polynôme prêt.
- $\alpha\beta$ appelé avec la fenêtre $[v, v+1]$ est optimal pour savoir si la valeur de la racine est supérieure ou égale à v.

$\alpha\beta$ avec une fenêtre d'appel $[v, v+1]$ peut être utilisé efficacement pour *tester* la valeur MiniMax d'un arbre de jeu.

```
Fonction Test(etat, v)                    ▷ Test si MiniMax(Etat) > v
    val ← NegEchecαβ(s, v, v + 1)
    Retourner (val > v)
Fin Fonction
```

$\alpha\beta$ pour tester la valeur MiniMax

Optimalité de $\alpha\beta$

- $\alpha\beta$ est optimal à ordre de parcours fixé, à un polynôme prêt.
- $\alpha\beta$ appelé avec la fenêtre $[v, v+1]$ est optimal pour savoir si la valeur de la racine est supérieure ou égale à v.

$\alpha\beta$ avec une fenêtre d'appel $[v, v+1]$ peut être utilisé efficacement pour *tester* la valeur MiniMax d'un arbre de jeu.

```
Fonction Test(etat, v)                    ▷ Test si MiniMax(Etat) > v
    val ← NegEchecαβ(s, v, v+1)
    Retourner (val > v)
Fin Fonction
```

$\alpha\beta$ pour tester la valeur MiniMax

Optimalité de $\alpha\beta$

- $\alpha\beta$ est optimal à ordre de parcours fixé, à un polynôme prêt.
- $\alpha\beta$ appelé avec la fenêtre $[v, v+1]$ est optimal pour savoir si la valeur de la racine est supérieure ou égale à v.

$\alpha\beta$ avec une fenêtre d'appel $[v, v+1]$ peut être utilisé efficacement pour *tester* la valeur MiniMax d'un arbre de jeu.

```
Fonction Test(etat, v)                    ▷ Test si MiniMax(Etat) > v
    val ← NegEchecαβ(s, v, v + 1)
    Retourner (val > v)
Fin Fonction
```

$\alpha\beta$ pour tester la valeur MiniMax

Optimalité de $\alpha\beta$

- $\alpha\beta$ est optimal à ordre de parcours fixé, à un polynôme prêt.
- $\alpha\beta$ appelé avec la fenêtre $[v, v+1]$ est optimal pour savoir si la valeur de la racine est supérieure ou égale à v.

$\alpha\beta$ avec une fenêtre d'appel $[v, v+1]$ peut être utilisé efficacement pour *tester* la valeur MiniMax d'un arbre de jeu.

Fonction $Test(etat, v)$ ▷ Test si $MiniMax(Etat) > v$

  $val \leftarrow NegEchec\alpha\beta(s, v, v+1)$

  Retourner $(val > v)$

Fin Fonction

$\alpha\beta$ avec information d'échec

Une réécriture plus informative de $\alpha\beta$

Idée

Remonter et exploiter les raisons de l'échec de $\alpha\beta$. La nouvelle fonction va pouvoir remonter la meilleure valeur des fils quelles que soient les valeurs initiales de α et β.

Principes

On va utiliser les raisons de l'échec éventuel de $\alpha\beta$ pour savoir si on a visé juste lorsqu'on fait une recherche avec une fenêtre initiale réduite.

$\alpha\beta$ avec information d'échec

Une réécriture plus informative de $\alpha\beta$

Idée

Remonter et exploiter les raisons de l'échec de $\alpha\beta$. La nouvelle fonction va pouvoir remonter la meilleure valeur des fils quelles que soient les valeurs initiales de α et β.

Principes

On va utiliser les raisons de l'échec éventuel de $\alpha\beta$ pour savoir si on a visé juste lorsqu'on fait une recherche avec une fenêtre initiale réduite.

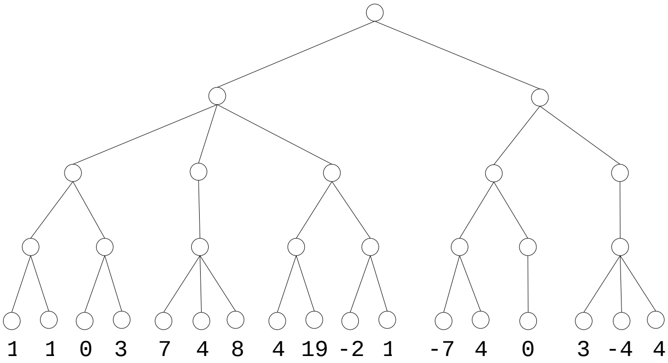
$Neg\alpha\beta$ avec information d'échec

Fonction $NegEchec\alpha\beta(etat, \alpha, \beta)$ ▷ Évaluation niveau AMI

- **Si** $EstFeuille(etat)$ **Alors** **Retourner** $evalue(etat)$
- **Fin Si**
- $Meilleur \leftarrow -\infty$
- **Pour Tout** successeur s de $etat$ **Faire**
  - $val \leftarrow -NegEchec\alpha\beta(s, -\beta, -\alpha)$
  - **Si** $val > Meilleur$ **Alors**
    - $Meilleur \leftarrow val$ ▷ meilleur choix local
    - **Si** $Meilleur > \alpha$ **Alors**
      - $\alpha \leftarrow Meilleur$
      - **Si** $\alpha \geq \beta$ **Alors** **Retourner** $Meilleur$ ▷ Coupe
      - **Fin Si**
    - **Fin Si**
  - **Fin Si**
- **Fin Pour**
- **Retourner** $Meilleur$ ▷ À la place de α

Fin Fonction

Exemple $\alpha\beta$ avec information de coupe



Utiliser les fenêtres et $NegEchec\alpha\beta$

Idée

Seulement vérifier, à chaque niveau, que les meilleurs coups sont le plus à gauche. Si ce n'est pas le cas, recalculer la nouvelle meilleure valeur.

```
Fonction Pαβ(etat)                                   ▷ Évaluation niveau AMI
    Si EstFeuille(etat) Alors Retourner evalue(etat)
    Fin Si
    Meilleur ← −Pαβ(fils ainé de etat)
    Pour Tout fils cadet s de etat Faire
        val ← −NegEchecαβ(s, −Meilleur, −(Meilleur − 1))
        Si val ≥ Meilleur Alors
            Meilleur ← −NegEchecαβ(s, −∞, −val)
        Fin Si
    Fin Pour
    Retourner Meilleur
Fin Fonction
```

Utiliser les fenêtres et $NegEchec\alpha\beta$

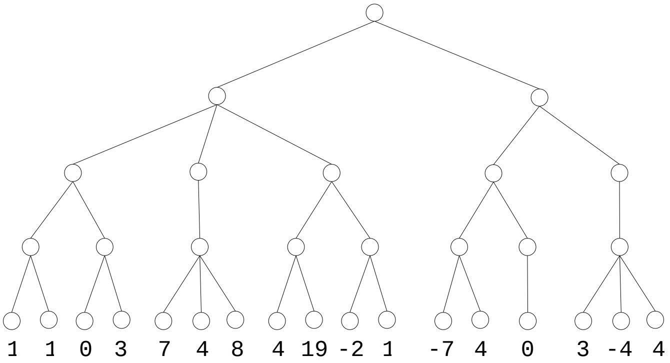
Idée

Seulement vérifier, à chaque niveau, que les meilleurs coups sont le plus à gauche. Si ce n'est pas le cas, recalculer la nouvelle meilleure valeur.

Fonction $P\alpha\beta(etat)$ ▷ Évaluation niveau AMI

    Si $EstFeuille(etat)$ **Alors** **Retourner** $evalue(etat)$
    Fin Si
    $Meilleur \leftarrow -P\alpha\beta($fils ainé de $etat)$
    Pour Tout fils cadet s de $etat$ **Faire**
        $val \leftarrow -NegEchec\alpha\beta(s, -Meilleur, -(Meilleur - 1))$
        Si $val \geq Meilleur$ **Alors**
            $Meilleur \leftarrow -NegEchec\alpha\beta(s, -\infty, -val)$
        Fin Si
    Fin Pour
    Retourner $Meilleur$
Fin Fonction

Exemple de déroulement de P-$\alpha\beta$



Généralisation de P-$\alpha\beta$

Problèmes

P-$\alpha\beta$ est asymétrique : si la bonne valeur n'est pas sur la première branche, il va repasser beaucoup de temps à recalculer des valeurs MiniMax.

Extension

L'algorithme suivant :

- Ne limite pas l'appel de la fonction $Test$ aux fils les plus à gauche

Généralisation de P-$\alpha\beta$

Problèmes

P-$\alpha\beta$ est asymétrique : si la bonne valeur n'est pas sur la première branche, il va repasser beaucoup de temps à recalculer des valeurs MiniMax.

Extension

L'algorithme suivant :

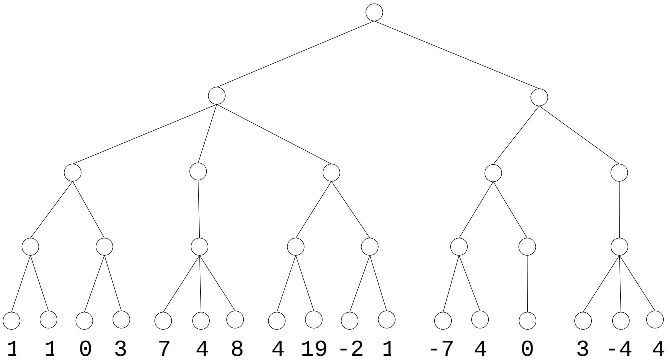
- Ne limite pas l'appel de la fonction $Test$ aux fils les plus à gauche

Scout, toujours

Fonction $Scout(etat)$ ▷ Évaluation niveau AMI

    Si $EstFeuille(etat)$ **Alors Retourner** $evalue(etat)$
    Fin Si
    $Meilleur \leftarrow -Scout($fils ainé de $etat)$ ▷ Le premier Meilleur
    Pour Tout fils cadet s de $etat$ **Faire**
        Si Non $Test(s, -Meilleur - 1)$ **Alors** ▷ Fils > Meilleur ?
            $Meilleur \leftarrow -Scout(s)$ ▷ Nouveau Meilleur
        Fin Si
    Fin Pour
    Retourner $Meilleur$
Fin Fonction

Exemple de déroulement de Scout



Résultats de Scout

Des études ont montré

Le comportement asymptotique de Scout est identique à $\alpha\beta$. Dans certains jeux, il peut lui être supérieur (jusqu'à 40% pour l'Awélé). Scout semble adapté aux arbres profonds, avec facteur de branchement faible.

D'autres formulations plus efficaces ont été données : PVS et NegaScout.

NegaScout

NegaScout domine $\alpha\beta$: il n'explore jamais un noeud qui serait coupé par $\alpha\beta$. L'inverse n'est pas vrai.

Résultats de Scout

Des études ont montré

Le comportement asymptotique de Scout est identique à $\alpha\beta$. Dans certains jeux, il peut lui être supérieur (jusqu'à 40% pour l'Awélé). Scout semble adapté aux arbres profonds, avec facteur de branchement faible.

D'autres formulations plus efficaces ont été données : PVS et NegaScout.

NegaScout

NegaScout domine $\alpha\beta$: il n'explore jamais un noeud qui serait coupé par $\alpha\beta$. L'inverse n'est pas vrai.

Résultats de Scout

Des études ont montré

Le comportement asymptotique de Scout est identique à $\alpha\beta$. Dans certains jeux, il peut lui être supérieur (jusqu'à 40% pour l'Awélé). Scout semble adapté aux arbres profonds, avec facteur de branchement faible.

D'autres formulations plus efficaces ont été données : PVS et NegaScout.

NegaScout

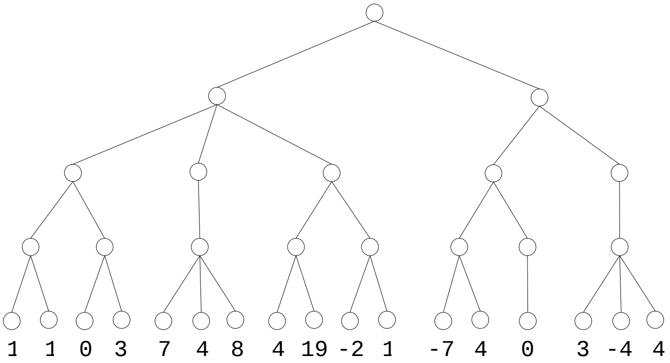
NegaScout domine $\alpha\beta$: il n'explore jamais un noeud qui serait coupé par $\alpha\beta$. L'inverse n'est pas vrai.

NegaScout

Fonction $NegaScout(etat, \alpha, \beta)$ ▷ Évaluation niveau AMI

  Si $EstFeuille(etat)$ **Alors** **Retourner** $evalue(etat)$
  Fin Si
  $\beta' \leftarrow \beta$
  Pour Tout fils s de $etat$ **Faire**
    $val \leftarrow -NegaScout(s, -\beta', -\alpha)$
    Si $(\alpha < val < \beta)$ Et $Cadet(etat)$ Et Non $EstFeuille(s)$ **Alors**
      $\alpha \leftarrow -NegaScout(s, -\beta, -val)$ ▷ Nouvelle recherche
    Fin Si
    $\alpha \leftarrow max(\alpha, val)$
    Si $\alpha \geq \beta$ **Alors** **Retourner** α ▷ Coupe
    Fin Si
    $\beta' \leftarrow \alpha + 1$ ▷ Nouvelle fenêtre de recherche
  Fin Pour
  Retourner α
Fin Fonction

Exemple de déroulement de NegaScout



NegaC* : Principes

NegaC* pousse encore l'idée de l'utilisation des fenêtres minimales.

Idée

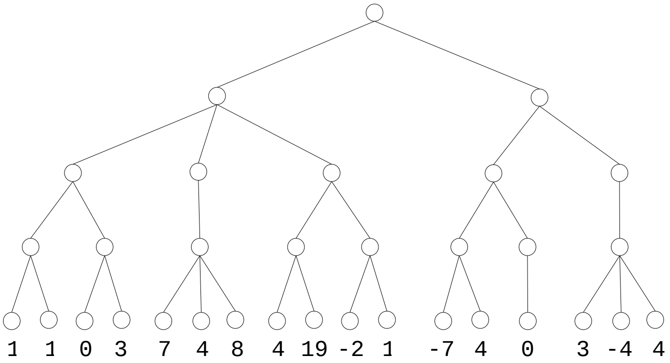
Faire une recherche dichotomique de la valeur minimax de l'arbre en utilisant le test sur les fenêtres minimales.

NegaC*

Fonction $NegaC*(etat)$ ▷ Évaluation niveau AMI

  Si $EstFeuille(etat)$ **Alors** **Retourner** $evalue(etat)$
  Fin Si
  $inf \leftarrow -\infty$
  $sup \leftarrow +\infty$
  Tant que $inf \neq sup$ **Faire**
    $v \leftarrow [(inf + sup)/2]$
    $t \leftarrow NegEchec\alpha\beta(etat, v, v+1)$
    Si $t > v$ **Alors**
      $inf \leftarrow t$
    Sinon
      $sup \leftarrow t$
    Fin Si
  Fin Tant que
  Retourner inf
Fin Fonction

Exemple de déroulement de NegaC*



$MTD(f)$: Idées

La recherche dichotomique est une bonne idée. Mais la valeur renvoyée par $NegEchec\alpha\beta$ peut encore permettre d'aller plus loin. Elle va permettre d'**encadrer** de plus en plus précisément la valeur MiniMax de l'arbre à chaque appel.

$MTD(f)$: Idées

La recherche dichotomique est une bonne idée. Mais la valeur renvoyée par $NegEchec\alpha\beta$ peut encore permettre d'aller plus loin. Elle va permettre d'**encadrer** de plus en plus précisément la valeur MiniMax de l'arbre à chaque appel.

$MTD(f)$: Idées

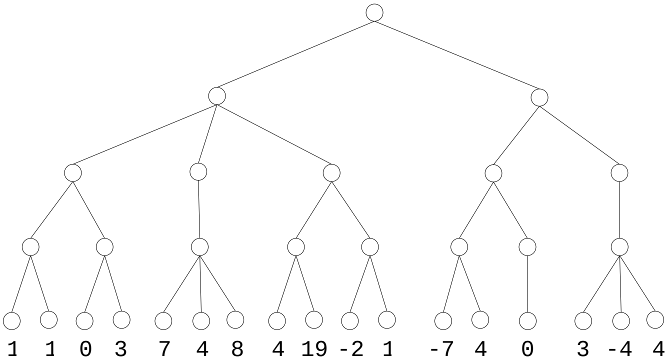
La recherche dichotomique est une bonne idée. Mais la valeur renvoyée par $NegEchec\alpha\beta$ peut encore permettre d'aller plus loin. Elle va permettre d'**encadrer** de plus en plus précisément la valeur MiniMax de l'arbre à chaque appel.

$MTD(f)$: le code

Fonction $MTD(etat, f)$

  $Supp \leftarrow +\infty$
  $Inf \leftarrow -\infty$
  Répéter
    Si $f = Inf$ **Alors** $\beta \leftarrow f + 1$
    Sinon $\beta \leftarrow f$
    Fin Si
    $f \leftarrow NegEchec\alpha\beta(etat, \beta - 1, \beta)$
    Si $f < \beta$ **Alors** $Supp \leftarrow f$
    Sinon $Inf \leftarrow f$
    Fin Si
  Jusqu'à $Supp \geq Inf$
  Retourner f
Fin Fonction

Exemple de déroulement de $MTD(f)$



Synthèse des algos avec fenêtre réduite

L'ordre est primordial !

Cette famille d'algorithmes suppose qu'un bon ordre est donné pour le développement des fils.

Pour $MTD(f)$ on va recommencer plusieurs fois la même recherche. Il faut impérativement utiliser les techniques de mémorisation des meilleurs coups.
Généralement, tous ces algos utilisent une mémoire pour stocker les espaces de jeu déjà vus, en les associant aux valeurs trouvées. C'est la clé de leurs performances.

Synthèse des algos avec fenêtre réduite

L'ordre est primordial !

Cette famille d'algorithmes suppose qu'un bon ordre est donné pour le développement des fils.

Pour $MTD(f)$ on va recommencer plusieurs fois la même recherche. Il faut impérativement utiliser les techniques de mémorisation des meilleurs coups.
Généralement, tous ces algos utilisent une mémoire pour stocker les espaces de jeu déjà vus, en les associant aux valeurs trouvées. C'est la clé de leurs performances.

Synthèse des algos avec fenêtre réduite

L'ordre est primordial !

Cette famille d'algorithmes suppose qu'un bon ordre est donné pour le développement des fils.

Pour $MTD(f)$ on va recommencer plusieurs fois la même recherche.
Il faut impérativement utiliser les techniques de mémorisation des meilleurs coups.
Généralement, tous ces algos utilisent une mémoire pour stocker les espaces de jeu déjà vus, en les associant aux valeurs trouvées. C'est la clé de leurs performances.

Petite conclusion

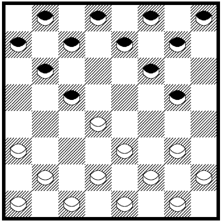
Synthèse expérimentale

Sur l'Othello

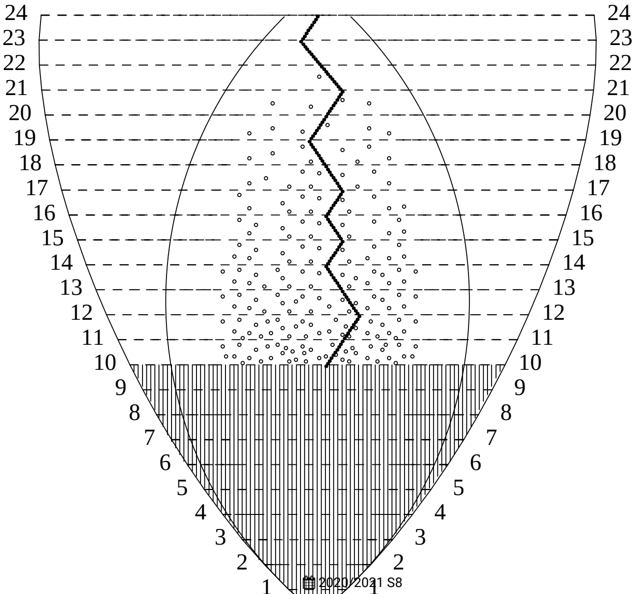
p	minimax	Scout	α-β	SSS*
1	3 – 0,37	4 – 0,45	3 – 0,36	3 – 0,36
2	14 – 1,38	21 – 2.09	13 – 1,4	11 – 1,19
3	61 – 6,0	45 – 5,0	37 – 3,9	35 – 3,7
4	349 – 32,5	246 – 23,7	150 – 14,77	95 – 10,0
5	2050 – 185,7	615 – 61,6	418 – 41,4	292 – 19,2
6	13773 – 1213,5	2680 – 254,5	1830 – 172,9	1617 – 117,3

Fermeture des Dames

L'un des prix IJCAI'05 a été donné à Schaeffer pour avoir notamment prouvé que l'ouverture aux dames ci-dessous n'était pas gagnante ni perdante :



Fermeture des Dames



Aujourd'hui les échecs

Alphazero a battu Stockfish (100-0)

en 2006 : Le champion du monde a perdu (4-2) contre Deep Fritz

- sur un dual core Xeon
- 8 Millions de positions par secondes
- profondeur de 17-18 (heuristiques).

Ce qui était vrai en 1960...

En guise de conclusion

[Samuel, 1960] « Just as it was impossible to begin the discussion of game-playing machines without referring to the hoaxes of the past, it is equally unthinkable to close the discussion without a diagnosis. Programming computers to play games is but one stage in the development of an understanding of the methods which must be employed for the machine simulation of intellectual behavior. As we progress in this understanding it seems reasonable to assume that these newer techniques will be applied to real-life situations with increasing frequency, and the effort devoted to games... will decrease. Perhaps we have not yet reached this turning point, and we may still have much to learn the study of games. »

Cette citation est toujours d'actualité... 55 ans après...