Algorithme min-max et élagage α - β

Quentin Fortier

January 25, 2026

Algorithme min-max

L'algorithme de calcul des attracteurs demande de parcourir chaque sommet du graphe des configurations possibles.

Il est donc beaucoup trop lent pour des jeux comme les échecs ($\approx 10^{44}$ sommets) ou le go ($\approx 10^{170}$) où le nombre de configurations est très grand.

Algorithme min-max

On rappelle qu'une heuristique est une fonction qui à une configuration associe une valeur dans $\mathbb{R}$ pour aider la recherche.

Algorithme min-max

On rappelle qu'une heuristique est une fonction qui à une configuration associe une valeur dans $\mathbb{R}$ pour aider la recherche.

Exemples :

- L'algorithme A^* utilise une heuristique pour estimer la distance entre un sommet et la destination.
- La méthode de branch-and-bound utilise une heuristique pour majorer la valeur d'une solution partielle.

Algorithme min-max

L'algorithme min-max utilise une heuristique h qui estime à quel point la configuration s est favorable à un joueur : plus $h(s)$ est grand, plus v est favorable au joueur 0 et inversement.
On prend en général $h(s) = \infty$ ($h(s) = -\infty$) si s est gagnant pour le joueur 0 (resp. 1).

Algorithme min-max

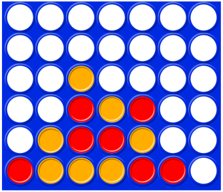
L'algorithme min-max utilise une heuristique h qui estime à quel point la configuration s est favorable à un joueur : plus $h(s)$ est grand, plus v est favorable au joueur 0 et inversement.
On prend en général $h(s) = \infty$ ($h(s) = -\infty$) si s est gagnant pour le joueur 0 (resp. 1).

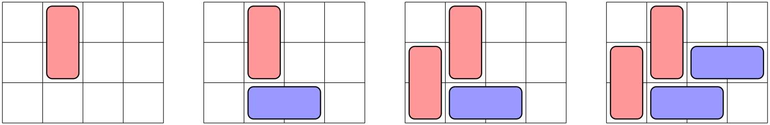
<u>Remarque</u> : Aussi bien pour l'algorithme A* que min-max, utiliser une heuristique permet d'accélerer la recherche, mais le résultat n'est pas forcément optimal. Par contre, branch-and-bound donne une solution optimale.

Algorithme min-max

Question

1. Proposer une heuristique pour le puissance 4.
2. Proposer une heuristique pour le domineering.

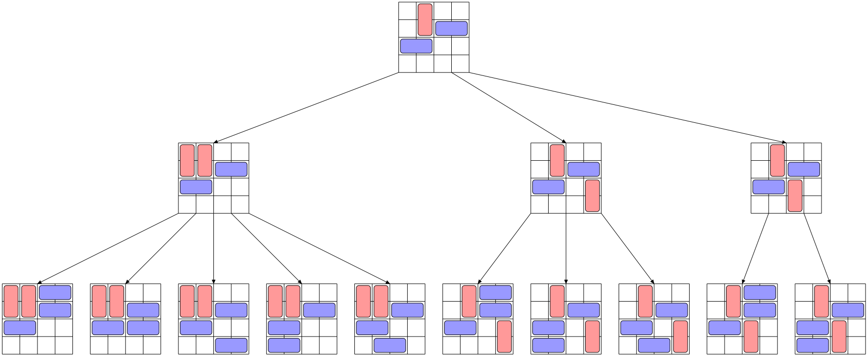




Algorithme min-max

On fixe une profondeur $p \in \mathbb{N}$.
L'algorithme min-max considère, depuis la position en cours, le graphe acyclique des positions atteignables après au plus p coups.

Exemple : graphe des positions atteignables après $p = 2$ coups.



Algorithme min-max

L'algorithme min-max donne une valeur à chaque sommet de l'arbre de proche en proche :

1. Calcul de l'heuristique des sommets à profondeur p et ceux sans successeurs.

Algorithme min-max

L'algorithme min-max donne une valeur à chaque sommet de l'arbre de proche en proche :

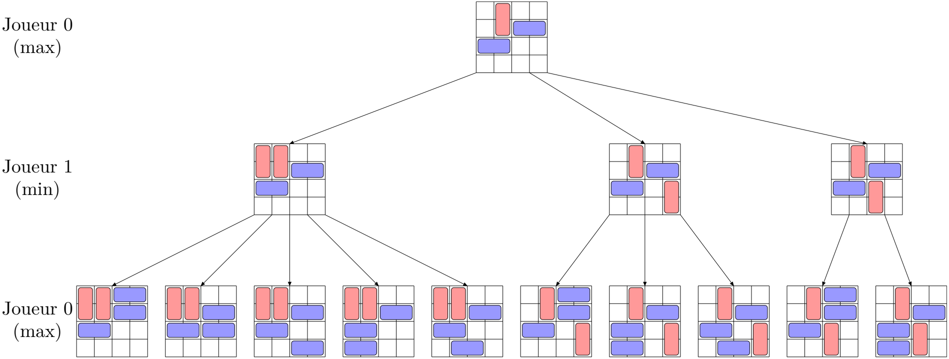
1. Calcul de l'heuristique des sommets à profondeur p et ceux sans successeurs.
2. Calcul de la valeur des sommets à profondeur $p - 1$ en prenant le maximum (pour le joueur 0) ou le minimum (pour le joueur 1) des valeurs des successeurs.

Algorithme min-max

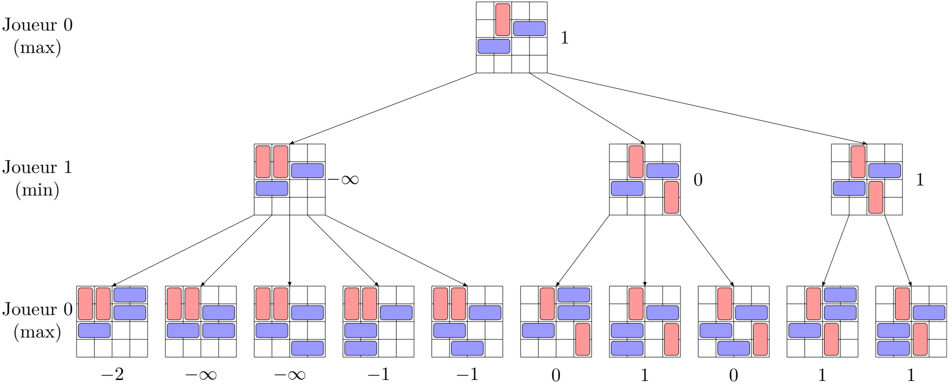
L'algorithme min-max donne une valeur à chaque sommet de l'arbre de proche en proche :

1. Calcul de l'heuristique des sommets à profondeur p et ceux sans successeurs.
2. Calcul de la valeur des sommets à profondeur $p - 1$ en prenant le maximum (pour le joueur 0) ou le minimum (pour le joueur 1) des valeurs des successeurs.
3. ...
4. Calcul de la valeur de la racine.

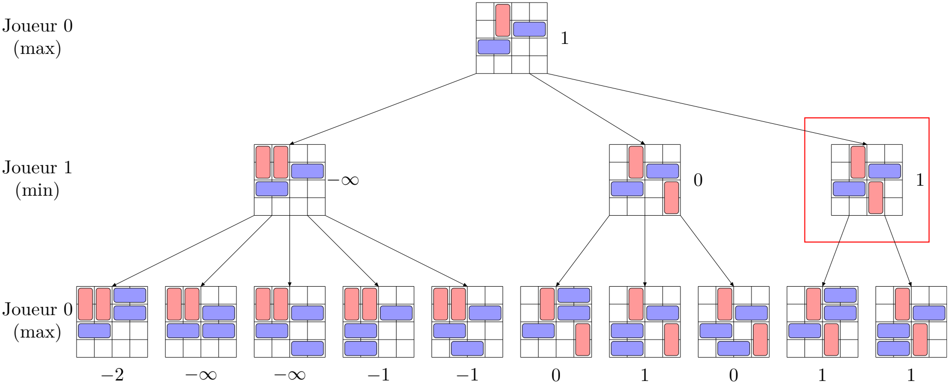
Algorithme min-max



Algorithme min-max



Algorithme min-max

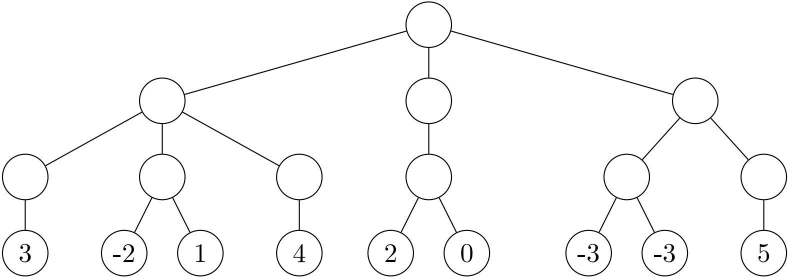


Le joueur 0 choisit le coup maximisant la valeur du successeur (le joueur 1 choisirait le coup minimisant l'heuristique).

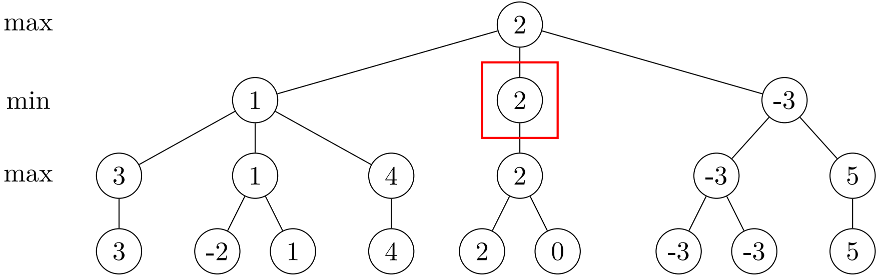
Algorithme min-max

Question

Compléter l'arbre min-max ci-dessous où on a mis les valeurs de l'heuristique à profondeur p. Le joueur qui joue en premier souhaite maximiser l'heuristique.



Algorithme min-max



Algorithme min-max

Question

Écrire une fonction récursive `minmax p j s` renvoyant la valeur de la configuration `s`, où `j` est le joueur actuel et `p` est la profondeur maximum. On suppose définie une heuristique `h` et une fonction `successeur` donnant la liste des configurations atteignables depuis une configuration donnée.

Algorithme min-max

```
let rec minmax p j s =
    let succ = List.map (minmax (p - 1) (1 - j)) (successeurs s) i
    if succ = [] || p = 0 then h s
    else if j = 0 then max_list succ
    else min_list succ
```

Remarque : On peut renvoyer un couple `(valeur, coup)` pour obtenir le prochain coup à jouer.

Élagage α - β

On peut accélerer l'algorithme min-max :

- En mémoïsant les configurations déjà rencontrées.
- En élaguant les branches inutiles.

Élagage α - β

L'élagage α - β conserve des bornes $\alpha \leq \beta$ (initialisées à $-\infty$ et ∞) encadrant la valeur de la racine, ce qui permet d'élaguer des branches inutiles :

Élagage α - β

L'élagage α - β conserve des bornes $\alpha \leq \beta$ (initialisées à $-\infty$ et ∞) encadrant la valeur de la racine, ce qui permet d'élaguer des branches inutiles :

- Si on est sur un sommet « max » et qu'on trouve une valeur supérieure à β, on peut arrêter le parcours (toutes les valeurs suivantes seront supérieures à β).
- Si on est sur un sommet « min » et qu'on trouve une valeur inférieure à α, on peut arrêter le parcours (toutes les valeurs suivantes seront inférieures à α).

Élagage α - β

L'élagage α - β conserve des bornes $\alpha \leq \beta$ (initialisées à $-\infty$ et ∞) encadrant la valeur de la racine, ce qui permet d'élaguer des branches inutiles :

- Si on est sur un sommet « max » et qu'on trouve une valeur supérieure à β, on peut arrêter le parcours (toutes les valeurs suivantes seront supérieures à β).
- Si on est sur un sommet « min » et qu'on trouve une valeur inférieure à α, on peut arrêter le parcours (toutes les valeurs suivantes seront inférieures à α).

On met à jour α et β au cours des appels récursifs :

- Si on est sur un sommet « max » et qu'on trouve une valeur v supérieure à α, on met v dans α (la valeur qui sera renvoyée sera au moins v).
- Si on est sur un sommet « min » et qu'on trouve une valeur v inférieure à β, on met v dans β (la valeur qui sera renvoyée sera

```python
def minmax(alpha, beta, p, s, j):
    succ = successeurs(s)
    if p == 0 or succ == []:
        return h(s)
    if j == 0:
        maxi = float('-inf')
        for t in succ:
            maxi = max(maxi, minmax(alpha, beta, p - 1, t, 1 - j))
            if maxi >= beta:
                return maxi
            alpha = max(alpha, maxi)
        return maxi
    else:
        mini = float('inf')
        for t in succ:
            mini = min(mini, minmax(alpha, beta, p - 1, t, 1 - j))
            if mini <= alpha:
                return mini
            beta = min(beta, mini)
        return mini
```

Élagage α - β

Exercice

Compléter l'arbre ci-dessous en utilisant l'algorithme min-max avec élagage α - β. Préciser les sommets élagués.