Intelligence artificielle

Programmation des jeux de reflexion

Elise Bonzon

elise.bonzon@u-paris.fr

LIPADE - Université de Paris
http://www.math-info.univ-paris5.fr/∽bonzon/

Programmation des jeux *vs.* résolution de problèmes

- L'adversaire est imprévisible $\Rightarrow$ la solution doit le prendre en compte

Programmation des jeux *vs.* résolution de problèmes

- L'adversaire est imprévisible $\Rightarrow$ la solution doit le prendre en compte
- Un temps limite est imposé $\Rightarrow$ quand il n'est pas possible d'atteindre le but il faut être capable de l'approximer

Programmation des jeux *vs.* résolution de problèmes

- L'adversaire est imprévisible $\Rightarrow$ la solution doit le prendre en compte
- Un temps limite est imposé $\Rightarrow$ quand il n'est pas possible d'atteindre le but il faut être capable de l'approximer
- Pistes étudiées :
  - algorithme pour joueur parfait (Von Neumann, 1944)
  - horizon fini, évaluation approximative (Zuse, 1945 ; Shannon 1950)
  - élagage pour réduire le coût de recherche (McCarthy, 1956)

Différents types de jeux

Information	Déterministe	Hasard
Parfaite		
Imparfaite		

Différents types de jeux

Information	Déterministe	Hasard
Parfaite	échecs, reversi, go, ...	
Imparfaite		

Différents types de jeux

Information	Déterministe	Hasard
Parfaite	échecs, reversi, go, ...	backgammon, monopoli, ...
Imparfaite		

Différents types de jeux

Information	**Déterministe**	**Hasard**
Parfaite	échecs, reversi, go, ...	backgammon, monopoli, ...
Imparfaite	bataille navale, mastermind, ...	

Différents types de jeux

Information	Déterministe	Hasard
Parfaite	échecs, reversi, go, ...	backgammon, monopoli, ...
Imparfaite	bataille navale, mastermind, ...	bridge, poker, scrabble, ...

Définition d'un jeu

Un jeu peut être formellement défini comme un problème de recherche, avec :

- S_0 : l'état initial, qui spécifie l'état du jeu au début de la partie
- `Player`(s) : définit quel joueur doit jouer dans l'état s
- `Action`(s) : retourne l'ensemble d'actions possibles dans l'état s
- `Result`(s, a) : fonction de transition, qui définit quel est le résultat de l'action a dans un état s
- `Terminal-Test`(s) : test de terminaison. Vrai si le jeu est fini dans l'état s, faux sinon. Les états dans lesquels le jeu est terminé sont appelés états terminaux.
- `Utility`(s, p) : une fonction d'utilité associe une valeur numérique à chaque état terminal s pour un joueur p

Jeux à somme nulle

- Un jeu à somme nulle est un jeu pour lequel la somme des utilités de tous les joueurs est la même pour toutes les issues possible du jeu
- Le nom peut préter à confusion, jeu à *somme constante* serait plus approprié
- Par exemple, le jeu d'échecs est un jeu à somme nulle :
  - $0 + 1$
  - $1 + 0$
  - $\frac{1}{2} + \frac{1}{2}$

Programmation des jeux de reflexion

1. L'algorithme `Minimax`
2. L'élagage α-β
3. Les jeux déterministes en pratique
4. Jeux non-déterministes
5. Jeux à information imparfaite
6. Conclusion

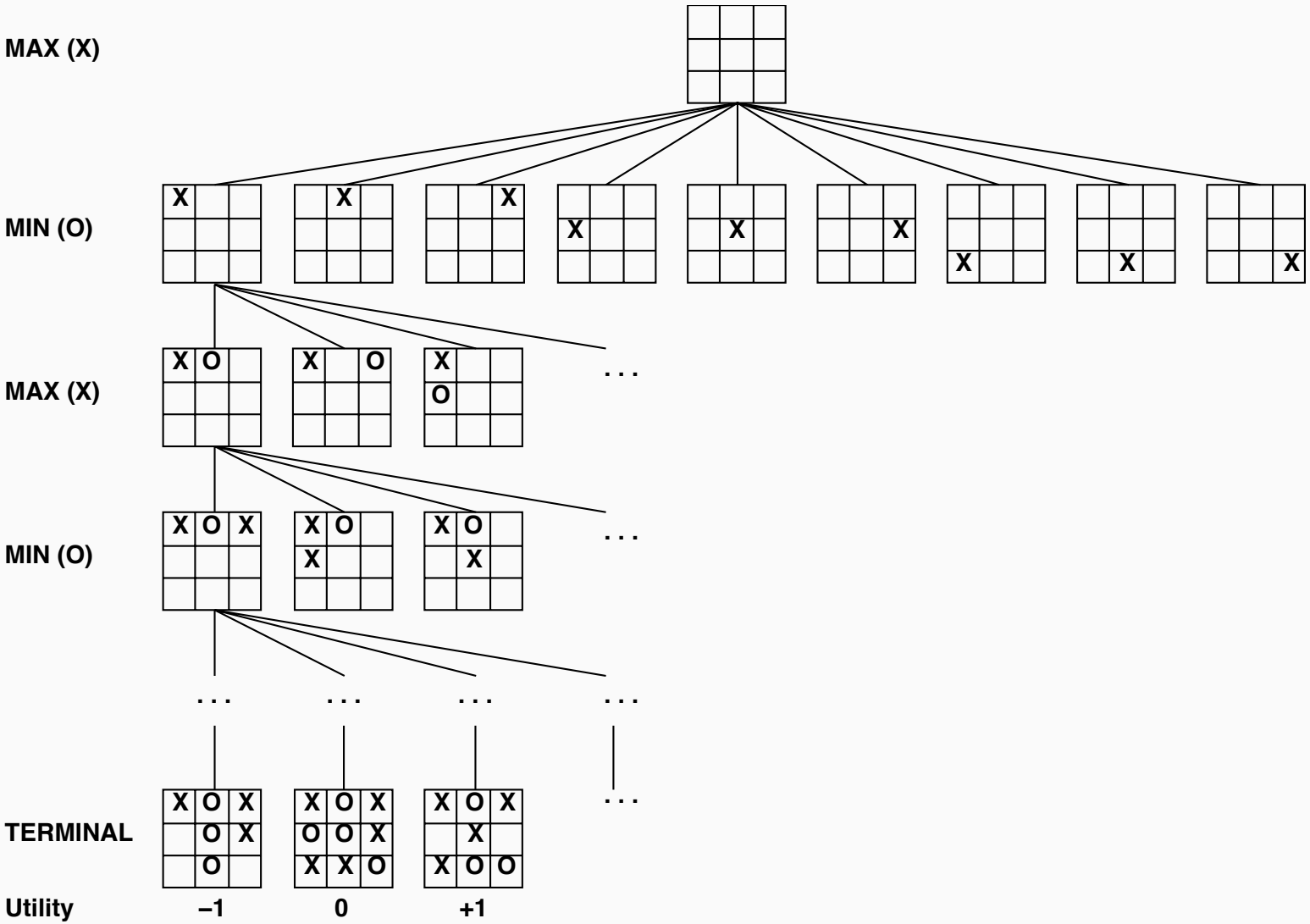
L'algorithme `Minimax`

L'algorithme `Minimax`

L'algorithme `Minimax` s'applique sur des jeux :

- à deux joueurs, appelés `Max` et `Min`
  - Par convention, `Max` joue en premier
- à somme nulle

Arbre de recherche du *tic-tac-toe*

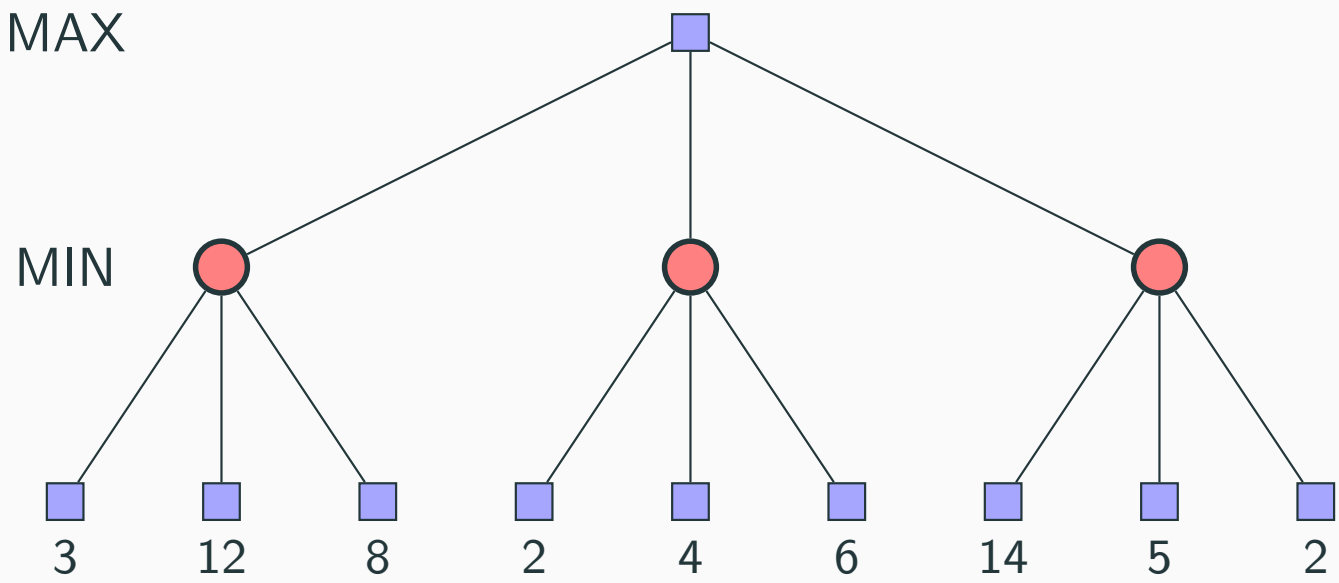


Algorithme de recherche `Minimax`

- Donne le coup parfait pour un jeu déterministe à information parfaite à somme nulle
- Idée : choisir le coup qui mène vers l'état qui a la meilleure valeur minimax = meilleure valeur possible contre le meilleur jeu de l'adversaire

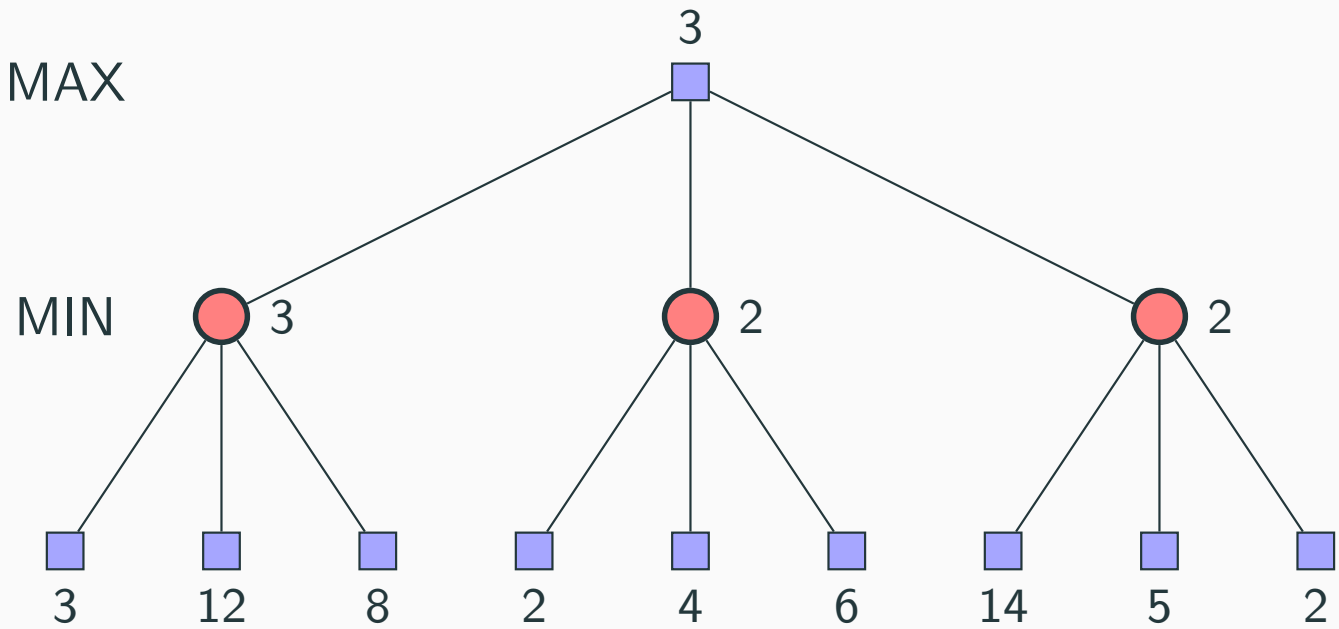
Algorithme de recherche `Minimax`

- Donne le coup parfait pour un jeu déterministe à information parfaite à somme nulle
- Idée : choisir le coup qui mène vers l'état qui a la meilleure valeur minimax = meilleure valeur possible contre le meilleur jeu de l'adversaire
- Exemple d'un jeu à deux coups :



Algorithme de recherche `Minimax`

- Donne le coup parfait pour un jeu déterministe à information parfaite à somme nulle
- Idée : choisir le coup qui mène vers l'état qui a la meilleure valeur minimax = meilleure valeur possible contre le meilleur jeu de l'adversaire
- Exemple d'un jeu à deux coups :



Algorithme de recherche `Minimax`

```
function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game
    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
    return v
```

Algorithme de recherche `Minimax`

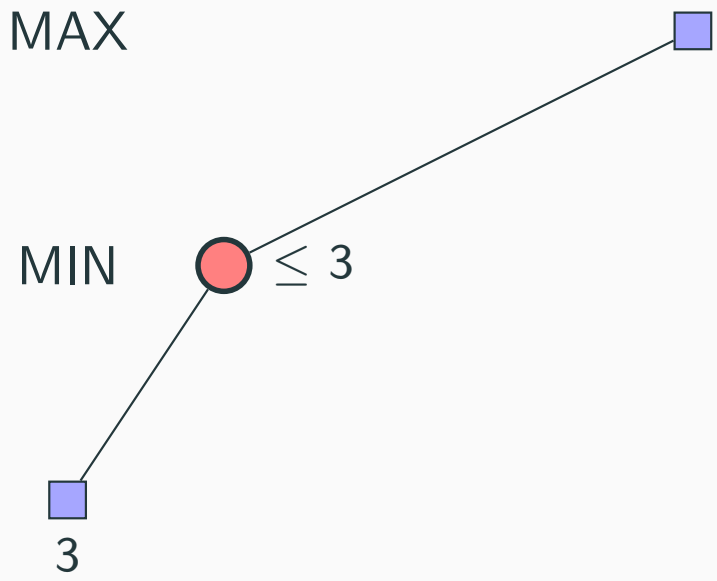
- Complet, si l'arbre est fini
- Optimal **si l'adversaire est optimal**
- Si b est le nombre maximal de coups possibles et m la profondeur maximale de l'arbre
  - Complexité en temps : $O(b^m)$
  - Complexité en espace : $O(bm)$
- Pour les échecs par exemple : $b \sim 35$, $m \sim 100 \Rightarrow$ solution exacte impossible

Algorithme de recherche `Minimax`

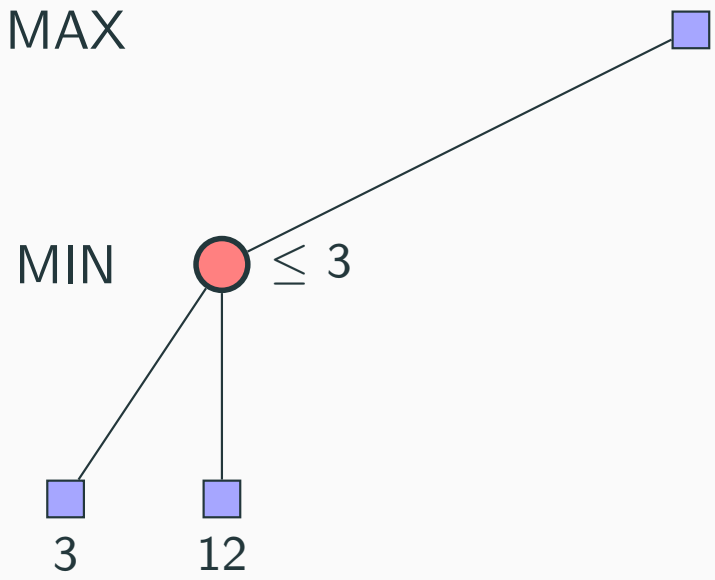
- Complet, si l'arbre est fini
- Optimal **si l'adversaire est optimal**
- Si b est le nombre maximal de coups possibles et m la profondeur maximale de l'arbre
  - Complexité en temps : $O(b^m)$
  - Complexité en espace : $O(bm)$
- Pour les échecs par exemple : $b \sim 35$, $m \sim 100 \Rightarrow$ solution exacte impossible
- Mais avons nous besoin d'explorer tous les chemins ?

L'élagage α-β

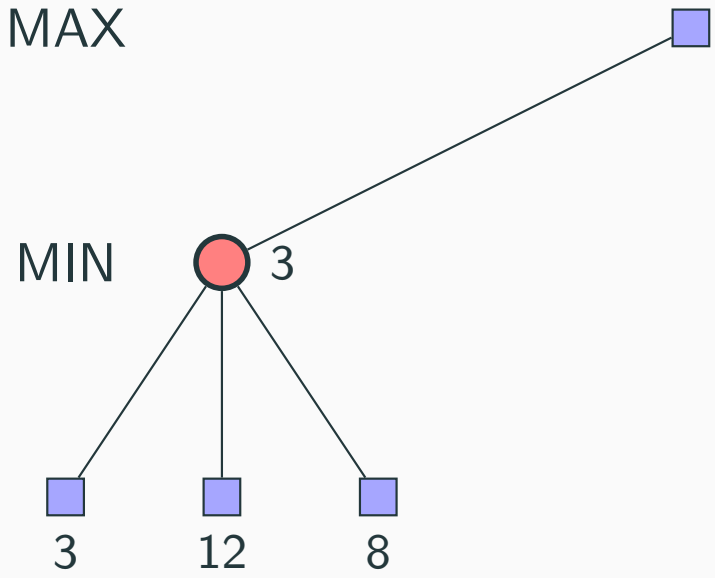
Exemple d'élagage α-β



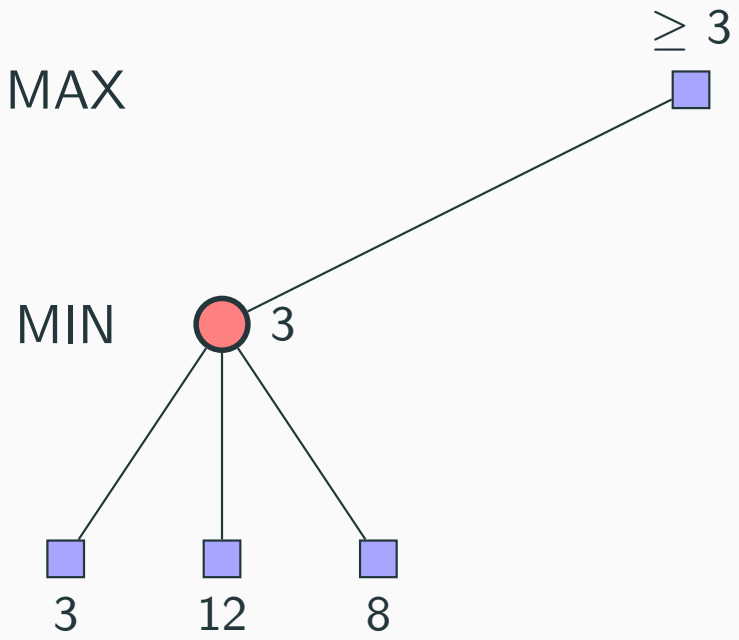
Exemple d'élagage α-β



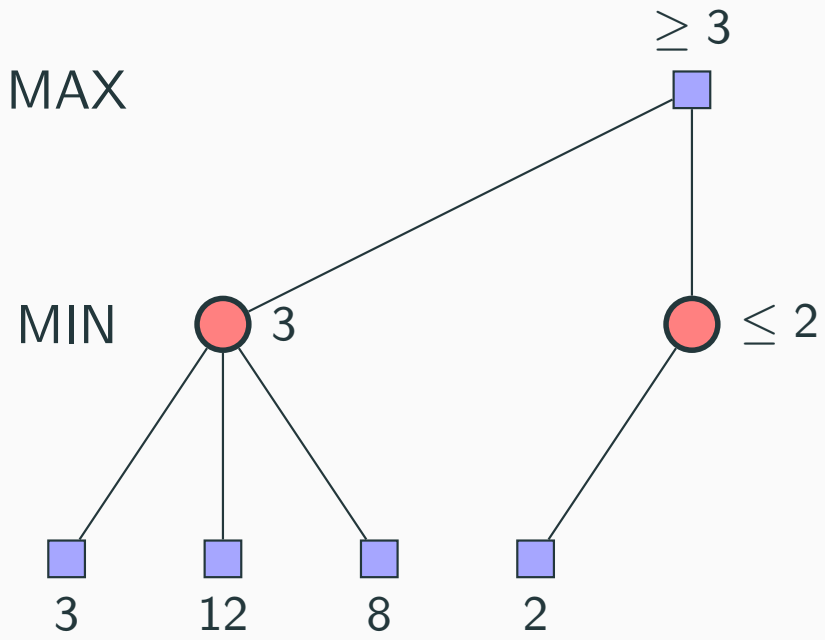
Exemple d'élagage α-β



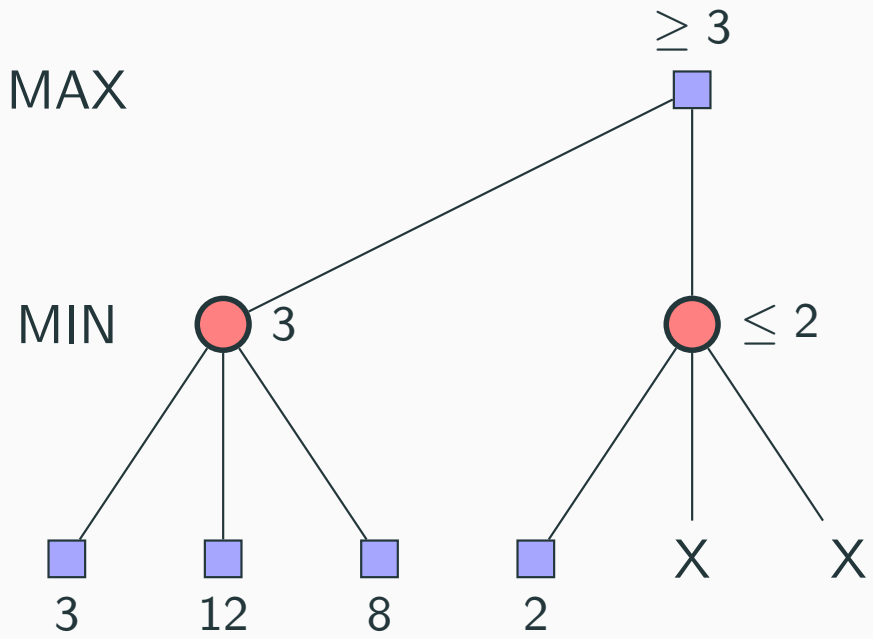
Exemple d'élagage α-β



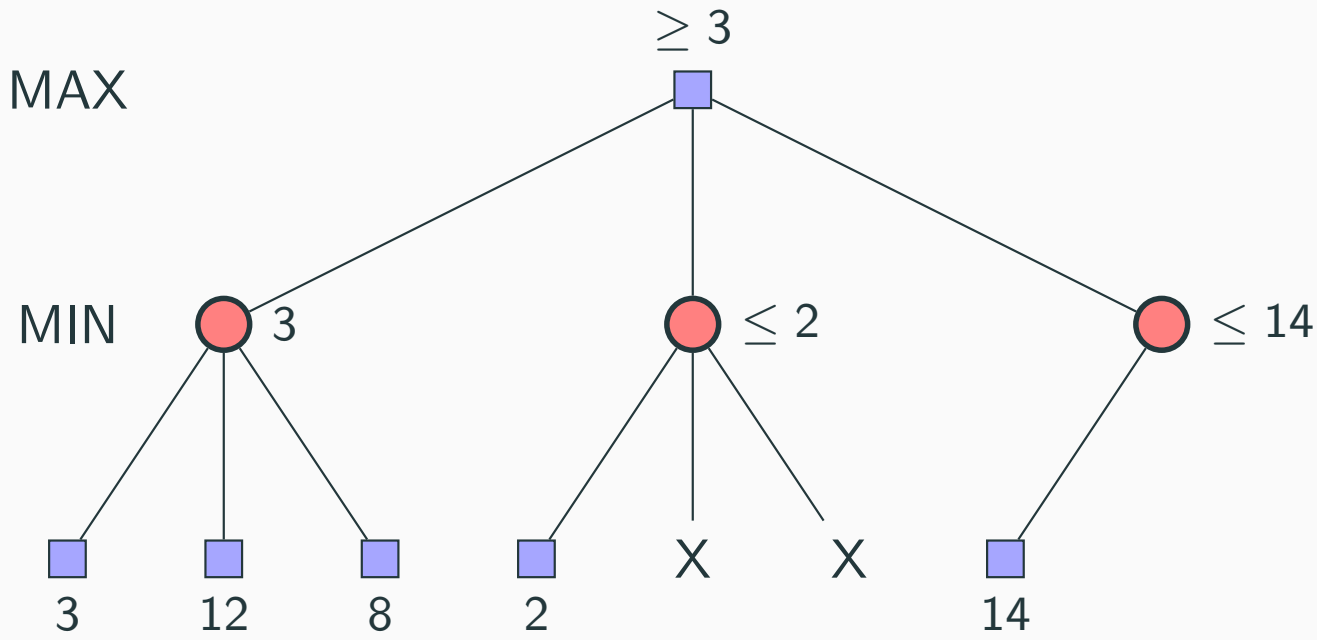
Exemple d'élagage α-β



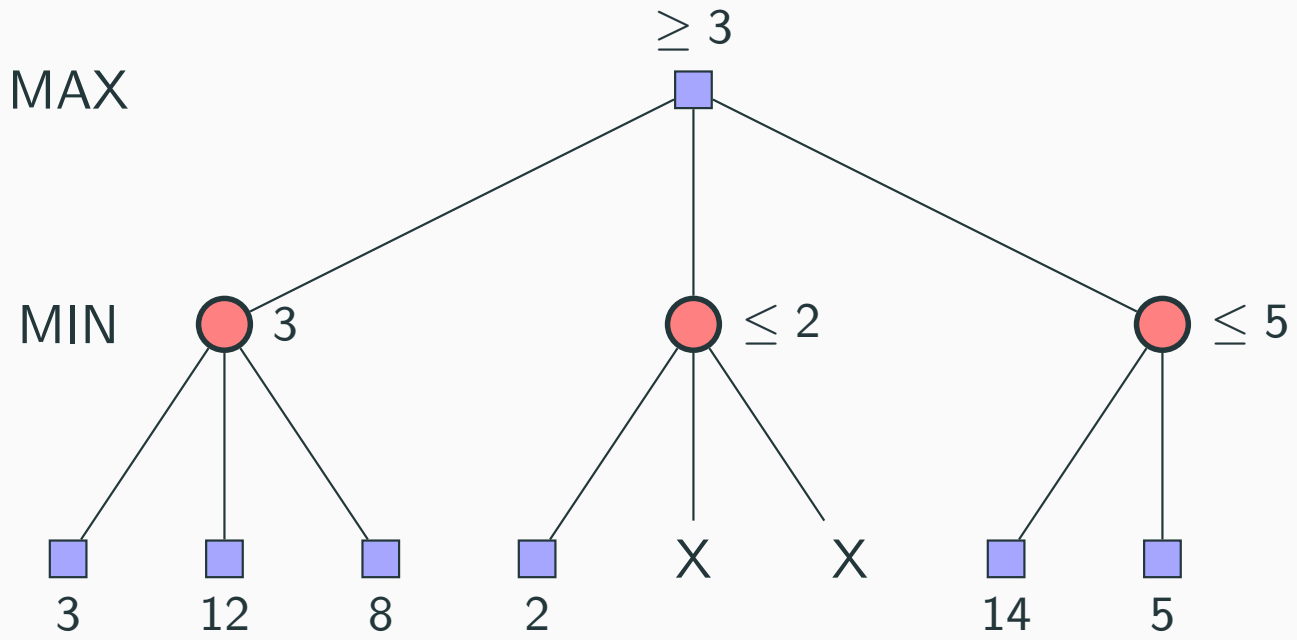
Exemple d'élagage α-β



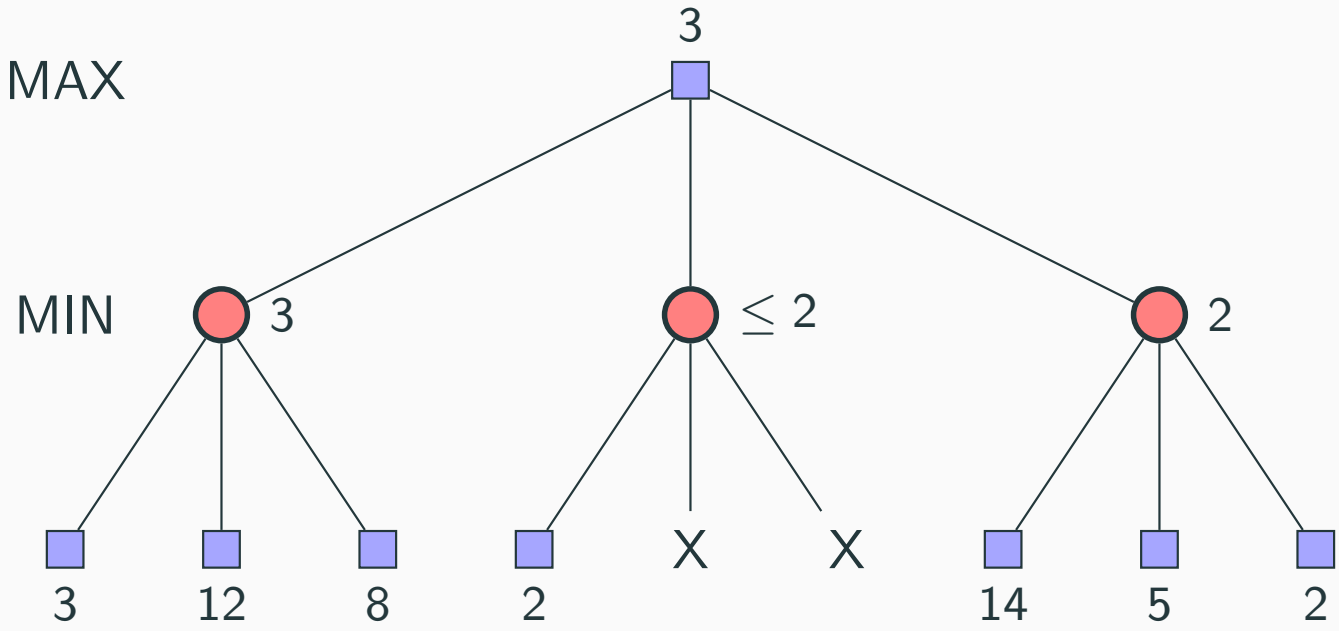
Exemple d'élagage α-β



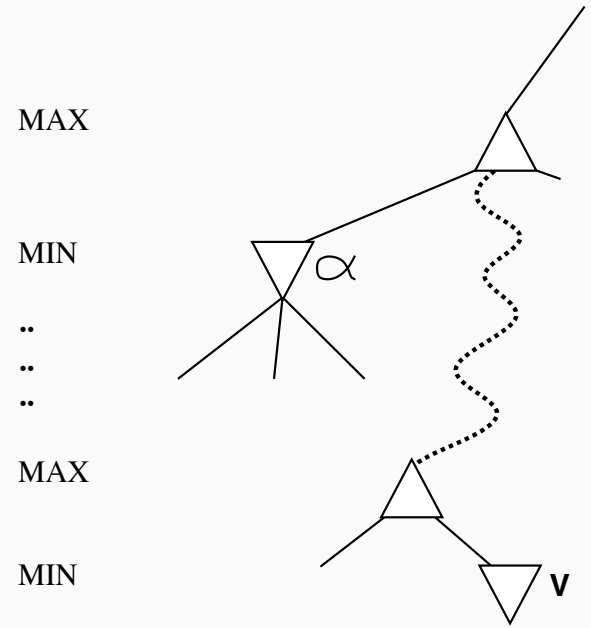
Exemple d'élagage α-β



Exemple d'élagage α-β



Pourquoi est-ce appelé α-β ?



- α est la meilleure valeur (la plus grande) pour `MAX` trouvée jusqu'à présent en dehors du chemin actuel
- Si V est pire que α, `MAX` va l'éviter $\Rightarrow$ élaguer la branche
- β définie similairement pour `MIN` : β est la meilleur valeur (la plus petite) pour `MIN` jusqu'à présent

Algorithme α-β

```
function ALPHA-BETA-DECISION(state) returns an action
    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s, α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v

function MIN-VALUE(state, α, β) returns a utility value
    same as MAX-VALUE but with roles of α, β reversed
```

Propriétés de α-β

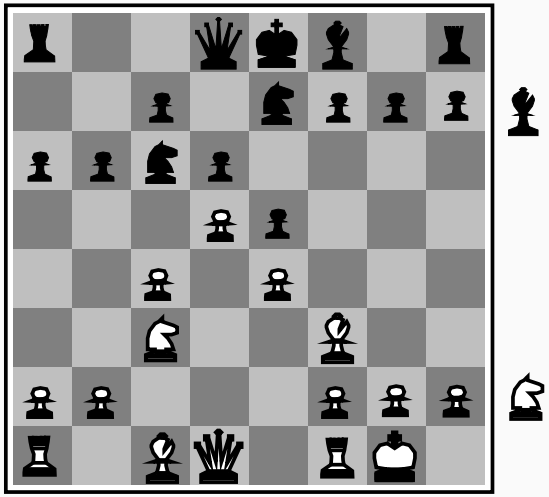
- L'élagage n'affecte pas le résultat final
- Un bon choix améliore l'efficacité de l'élagage
- Avec un "choix parfait" la complexité en temps est $O(b^{m/2})$
  - $\Rightarrow$ double la profondeur de recherche par rapport à `Minimax`
  - $\Rightarrow$ Mais trouver la solution exacte avec une complexité de l'ordre de 35^{50}, pour les échecs, est toujours impossible
- Par exemple, si on peut explorer 10^4 nœuds par seconde et que l'on a 100 secondes
  - $\Rightarrow$ On peut regarder 10^6 nœuds par coup $\sim 35^{8/2}$
  - $\Rightarrow$ α-β peut facilement atteindre des profondeurs de l'ordre de 8 coups d'avance aux échecs
  - $\Rightarrow$ plutôt bon programme d'échecs

Ressources limitées

Approches standard en cas de ressources limitées, ou d'espace de recherche trop grand :

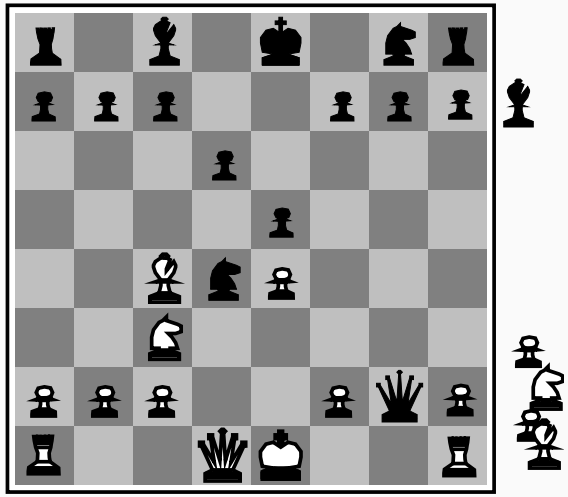
- Utiliser un test d'arrêt : remplacer `Terminal-Test` par `Cutoff-Test`
  - pour, par exemple, limiter la profondeur
- Utiliser une fonction d'évaluation : remplacer `Utility` par `Eval`
  - valeur estimée d'une position

Fonctions d'évaluation



Black to move

White slightly better

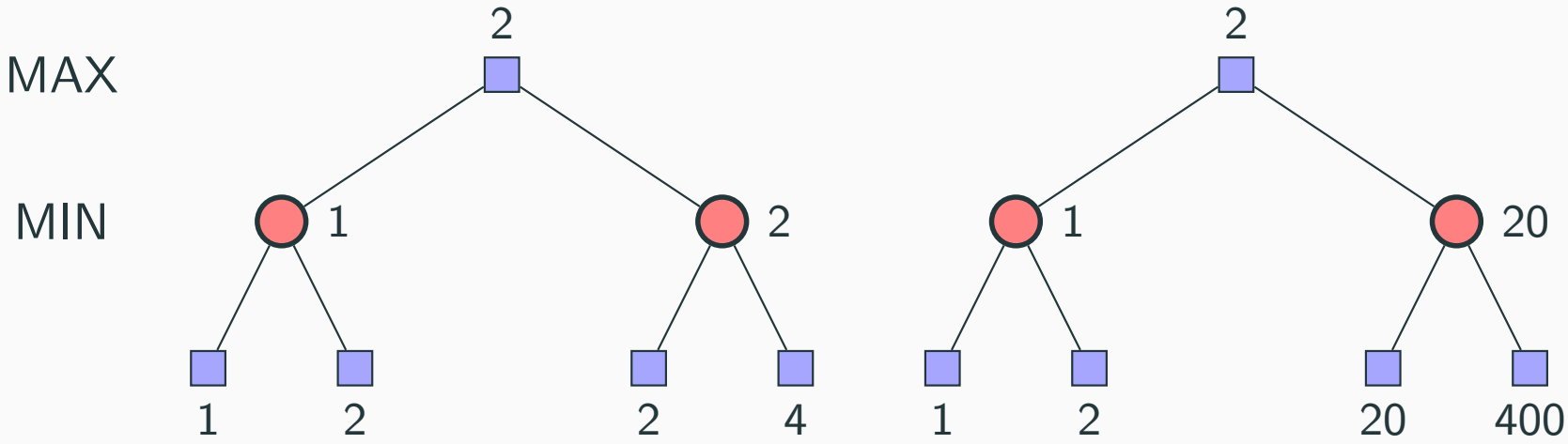


White to move

Black winning

- Aux échecs, on choisit par exemple une somme linéaire pondérée de caractéristiques
- $\texttt{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
- avec par exemple
  - $w_1 = 9$ et $f_1(s) =$ (nombre de reines blanches) - (nombre de reines noires),
  - etc.

α-β : la valeur exacte des nœuds *n'est pas* importante



- Seul l'ordre dans lequel on visite les nœuds est important
- Le comportement est préservé pour chaque transformation **monotone** de la fonction `Eval`
- Pour les jeux déterminisites, la fonction de résultat se comporte comme une **fonction d'utilité ordinale**

- L'ordre dans lequel on visite les fils est important
- Si on trouve rapidement une bonne valeur, on élague plus de nœuds
- On peut trier les fils par leur utilité
- D'autres améliorations sont possible
- Aux échecs on utilise au début du jeu des bases de données d'ouverture, au milieu α-β, et à la fin des algorithmes spéciaux

Les jeux déterministes en pratique

Quelques ordres de grandeur

- Tic-tac-toe : 3^9 états
- Rubik's cube : 10^{19} états
- Dames : 10^{40} états
- Echecs : 10^{120} états (facteur de branchement : 35)
- Go : 10^{172} états (facteur de branchement : > 300)

Les jeux déterministes en pratique

- **Puissance 4** : Le jeu est résolu : on a montré qu'il existait une stratégie gagnante pour le joueur qui commence à jouer. Cette stratégie étant stockée dans une base de données, il est facile pour un joueur artificiel de suivre cette stratégie et de gagner systématiquement.

Les jeux déterministes en pratique

- **Puissance 4** : Le jeu est résolu : on a montré qu'il existait une stratégie gagnante pour le joueur qui commence à jouer. Cette stratégie étant stockée dans une base de données, il est facile pour un joueur artificiel de suivre cette stratégie et de gagner systématiquement.
- **Dames** : Chinook a mis fin à un règne de 40 ans du champion du monde Marion Tinslay en 1994. Il a utilisé une base de données définissant un jeu parfait pour toutes les positions avec 8 pièces ou moins sur le damier. (Soit au total 443 748 401 247 positions)

Les jeux déterministes en pratique

- **Othello** : Les champions humains refusent de jouer contre les ordinateurs, qui sont trop bons

Les jeux déterministes en pratique

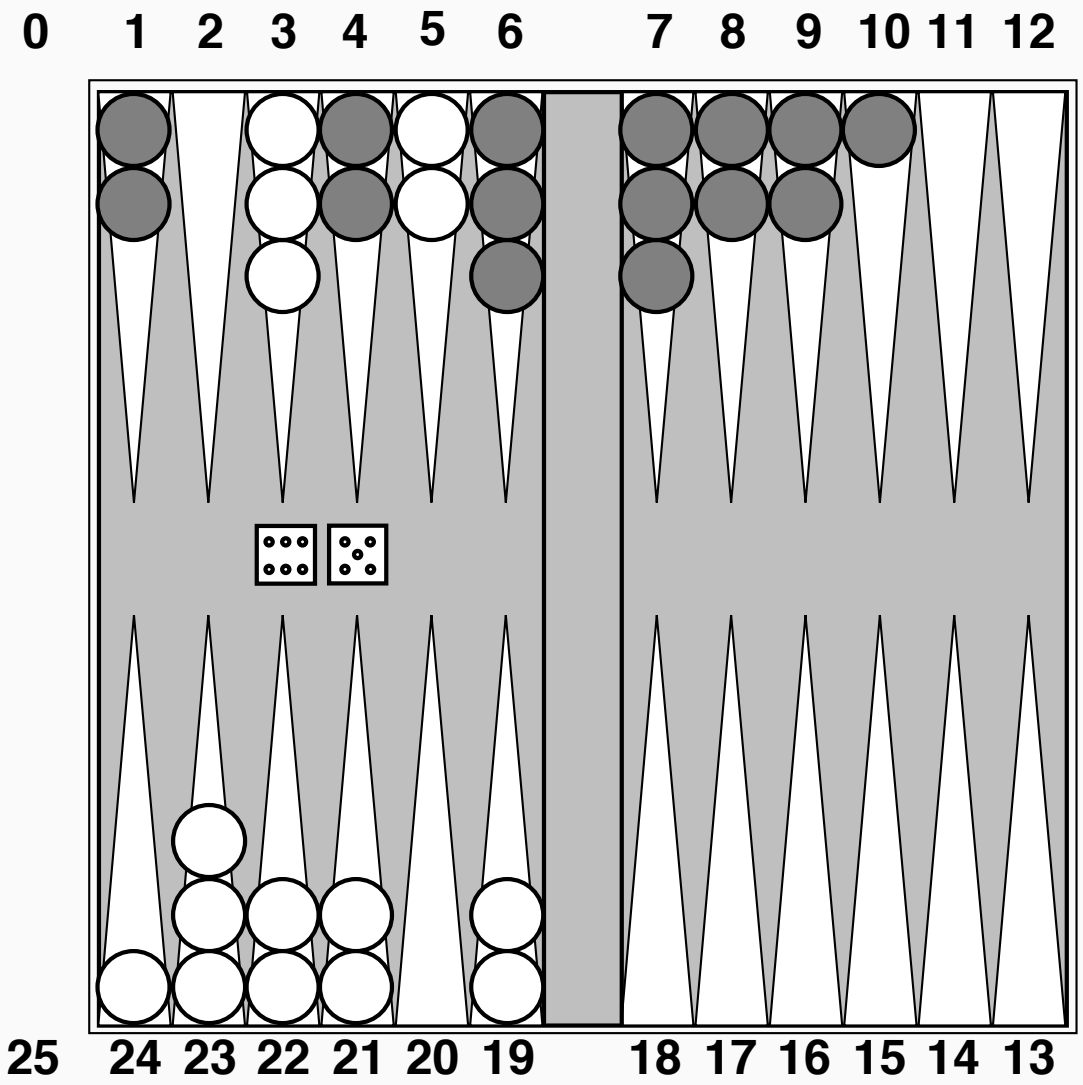
- **Othello** : Les champions humains refusent de jouer contre les ordinateurs, qui sont trop bons
- **Echecs** : Deep Blue a battu le champion du monde Gary Kasparov dans un match à 6 parties en 1997. Deep Blue cherche 200 million de positions par seconde, utilise une fonction d'évaluation très sophistiquée, et des méthodes non communiquées pour étendre quelques lignes de recherche jusqu'à 40 coups d'avance. En 2017, Stockfish est le meilleur logiciel d'échecs. Son niveau est largement supérieur à celui des meilleurs joueurs humains.

Les jeux déterministes en pratique

- **Go** :
  - $b > 300$, dans les années 90 la plupart des programmes utilisaient des bases de connaissances permettant de suggérer les coups les plus probables.
  - 2006, Monte-Carlo Tree Search (MCTS).
  - 2016, AlphaGo (Google Deepmind) a battu un joueur humain professionnel. AlphaGo utilise MCTS, des simulations et un réseau de neurones profond pour orienter la recherche. Le réseau de neurones a appris en utilisant des bases de coups joués par des joueurs humains professionnels.
  - 2017, AlphaGo Zéro confirme en battant un autre champion humain. Le réseau de neurones a appris en auto-jeu, sans utiliser de connaissances humaines.
  - Alpha Zéro apprend à jouer aux Echecs et bat Stockfish ; apprend à jouer au Shogi et bat le meilleur logiciel de Shogi. Approche générale pour les jeux déterministes à information complète.

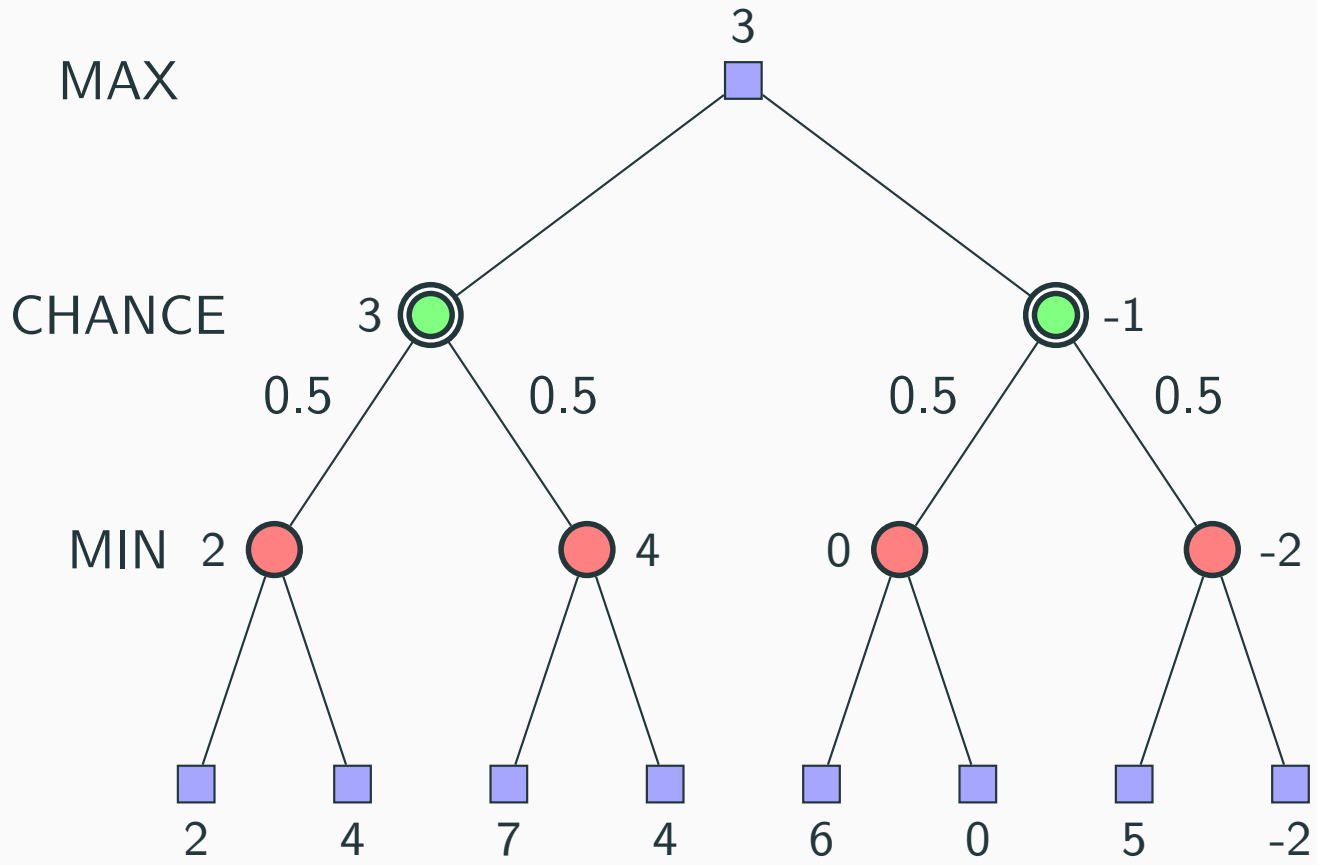
Jeux non-déterministes

Jeu non-déterministe : backgammon



Jeux non-déterministes

- Dans un jeu non-déterministe, la chance est introduite par un jeter de dés, une distribution de cartes ...
- Exemple simplifié avec le lancer d'une pièce de monnaie :

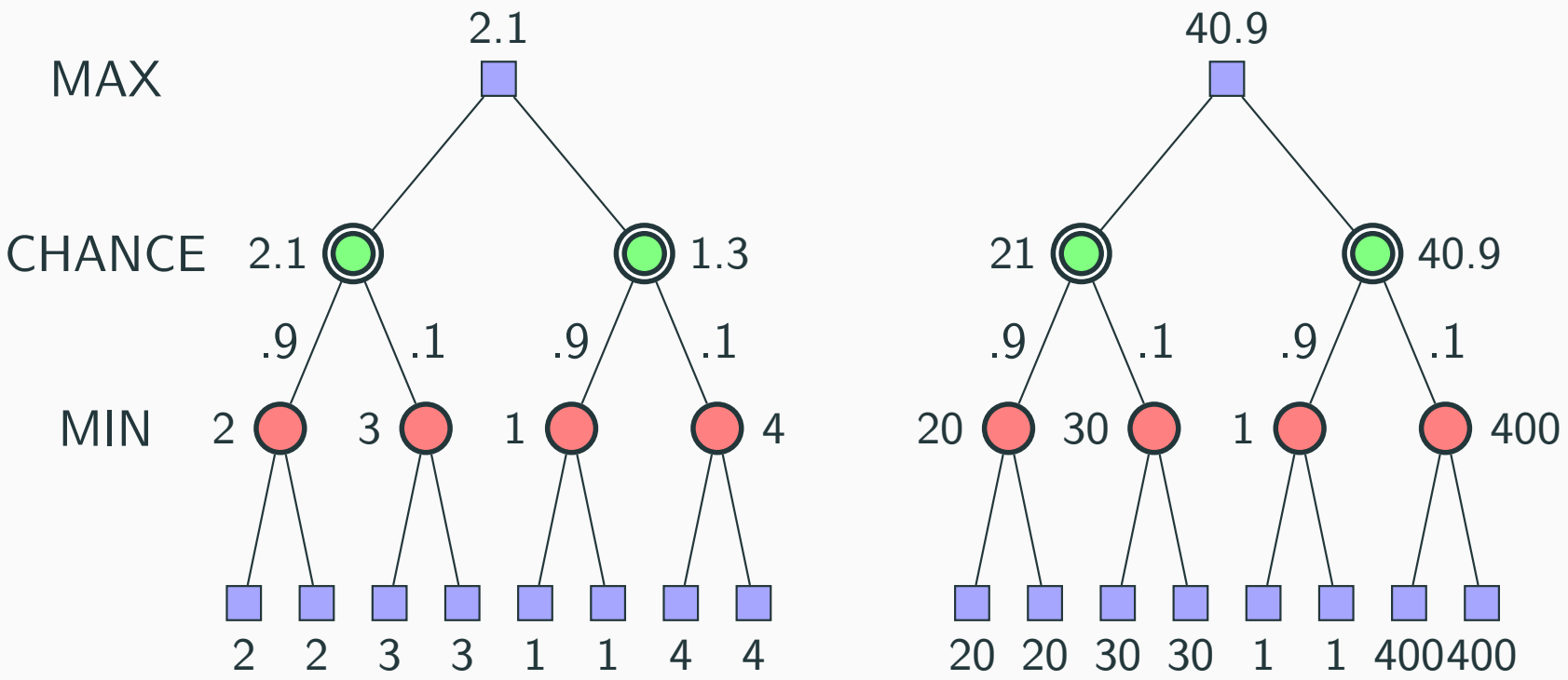


Algorithme pour les jeux non-déterministes

- `Expectiminimax` donne le coup parfait, comme `MiniMax`
- On doit considérer en plus les nœuds `Chance`
- On doit ajouter la ligne suivante à l'algorithme `MiniMax` : :

  if *state* is a `Max` node then
  return the highest `Expectiminimax-Value` of `Successors`(*state*)
  if *state* is a `Min` node then
  return the lowest `Expectiminimax-Value` of `Successors`(*state*)
  if *state* is a `Chance` node then
  return average `Expectiminimax-Value` of `Successors`(*state*)
- Si `Eval` est bornée, α-β peut-être adaptée

La valeur exacte des nœuds *est* importante



- Le comportement est préservé seulement pour chaque transformation **positive et linéaire** de la fonction `Eval`
- `Eval` doit être proportionnelle au gain attendu

Jeux à information imparfaite

Jeux à information imparfaite

- Par exemple, jeux de cartes où les cartes de l'adversaire ne sont pas connues
- Il est possible de calculer une probabilité pour chaque distribution de cartes possible
- Ressemble à un *gros dé* lancé au début du jeu
- Idée : calculer la valeur `Minimax` de chaque action dans chaque distribution possible, puis choisir l'action qui la valeur espérée maximale parmi toutes les distributions
- Cas spécial : si une action est optimale pour toutes les distributions, c'est une action optimale

Exemple de sens commun

- Jour 1
  - Route A mène à un petit tas de pièces d'or
  - Route B mène à une intersection
    - Si vous prenez à gauche, vous arriverez à une montagne de bijoux
    - Si vous prenez à droite, vous vous ferez écraser par un bus

Exemple de sens commun

- Jour 1
  - Route A mène à un petit tas de pièces d'or
  - Route B mène à une intersection
    - Si vous prenez à gauche, vous arriverez à une montagne de bijoux
    - Si vous prenez à droite, vous vous ferez écraser par un bus
- Jour 2
  - Route A mène à un petit tas de pièces d'or
  - Route B mène à une intersection
    - Si vous prenez à gauche, vous vous ferez écraser par un bus
    - Si vous prenez à droite, vous arriverez à une montagne de bijoux

Exemple de sens commun

- Jour 1
  - Route A mène à un petit tas de pièces d'or
  - Route B mène à une intersection
    - Si vous prenez à gauche, vous arriverez à une montagne de bijoux
    - Si vous prenez à droite, vous vous ferez écraser par un bus
- Jour 2
  - Route A mène à un petit tas de pièces d'or
  - Route B mène à une intersection
    - Si vous prenez à gauche, vous vous ferez écraser par un bus
    - Si vous prenez à droite, vous arriverez à une montagne de bijoux
- Jour 3
  - Route A mène à un petit tas de pièces d'or
  - Route B mène à une intersection
    - Devinez correctement et vous arriverez à une montagne de bijoux
    - Devinez incorrectement et vous vous ferez écraser par un bus

Jeux à information imparfaite

- L'idée que la valeur d'une actions est la moyenne de ses valeurs possibles est **fausse**
- Avec une observation partielle, la valeur d'une action dépend de l'état des connaissances ou l'état de croyance de l'agent
- Il est possible de générer un arbre de recherche sur les états de connaissances
- Mène à des comportements rationnels tels que :
  - Agir pour obtenir de l'information
  - Donner les informations que l'on a à son ou ses partenaire(s)
  - Agir au hasard pour minimiser les informations que l'on fournit à ses adversaires

Conclusion

Conclusion

- Etudier les jeux permet de soulever plusieurs points importants en IA
  - La perfection n'est pas atteignable $\Rightarrow$ nécessité d'approximer
  - Il faut penser à quoi penser
  - L'incertitude force à assigner des valeurs aux états
  - Les décisions optimales dépendent des informations sur l'état, pas de l'état réel
- Les jeux sont à l'IA ce que les grands prix sont à la conception des voitures

Et le projet dans tout ça ?

- Choisir un jeu assez difficile pour que ce soit intéressant
  - Le morpion, avec 3^9 états, peut être résolu avec un simple Minimax

Et le projet dans tout ça ?

- Choisir un jeu assez difficile pour que ce soit intéressant
  - Le morpion, avec 3^9 états, peut être résolu avec un simple Minimax
- Mais pas trop compliqué non plus...
  - Les règles du jeu d'échec sont très complexes à programmer !

Et le projet dans tout ça ?

- Choisir un jeu assez difficile pour que ce soit intéressant
  - Le morpion, avec 3^9 états, peut être résolu avec un simple Minimax
- Mais pas trop compliqué non plus...
  - Les règles du jeu d'échec sont très complexes à programmer !
- Méthodologie :

Et le projet dans tout ça ?

- Choisir un jeu assez difficile pour que ce soit intéressant
  - Le morpion, avec 3^9 états, peut être résolu avec un simple Minimax
- Mais pas trop compliqué non plus...
  - Les règles du jeu d'échec sont très complexes à programmer !
- Méthodologie :
  1. Définir la structure représentant les états (données + fonctions)

Et le projet dans tout ça ?

- Choisir un jeu assez difficile pour que ce soit intéressant
  - Le morpion, avec 3^9 états, peut être résolu avec un simple Minimax
- Mais pas trop compliqué non plus...
  - Les règles du jeu d'échec sont très complexes à programmer !
- Méthodologie :
  1. Définir la structure représentant les états (données + fonctions)
  2. Implémenter la boucle de jeu pour 2 joueurs humains

Et le projet dans tout ça ?

- Choisir un jeu assez difficile pour que ce soit intéressant
  - Le morpion, avec 3^9 états, peut être résolu avec un simple Minimax
- Mais pas trop compliqué non plus...
  - Les règles du jeu d'échec sont très complexes à programmer !
- Méthodologie :
  1. Définir la structure représentant les états (données + fonctions)
  2. Implémenter la boucle de jeu pour 2 joueurs humains
  3. Implémenter l'algorithme Minimax basique

Et le projet dans tout ça ?

- Choisir un jeu assez difficile pour que ce soit intéressant
  - Le morpion, avec 3^9 états, peut être résolu avec un simple Minimax
- Mais pas trop compliqué non plus...
  - Les règles du jeu d'échec sont très complexes à programmer !
- Méthodologie :
  1. Définir la structure représentant les états (données + fonctions)
  2. Implémenter la boucle de jeu pour 2 joueurs humains
  3. Implémenter l'algorithme Minimax basique
  4. Remplacer un joueur humain par l'ordinateur avec l'algorithme Minimax

Et le projet dans tout ça ?

- Choisir un jeu assez difficile pour que ce soit intéressant
  - Le morpion, avec 3^9 états, peut être résolu avec un simple Minimax
- Mais pas trop compliqué non plus...
  - Les règles du jeu d'échec sont très complexes à programmer !
- Méthodologie :
  1. Définir la structure représentant les états (données + fonctions)
  2. Implémenter la boucle de jeu pour 2 joueurs humains
  3. Implémenter l'algorithme Minimax basique
  4. Remplacer un joueur humain par l'ordinateur avec l'algorithme Minimax
  5. Implémenter et intégrer l'élagage $\alpha\beta$

Et le projet dans tout ça ?

- Choisir un jeu assez difficile pour que ce soit intéressant
  - Le morpion, avec 3^9 états, peut être résolu avec un simple Minimax
- Mais pas trop compliqué non plus...
  - Les règles du jeu d'échec sont très complexes à programmer !
- Méthodologie :
  1. Définir la structure représentant les états (données + fonctions)
  2. Implémenter la boucle de jeu pour 2 joueurs humains
  3. Implémenter l'algorithme Minimax basique
  4. Remplacer un joueur humain par l'ordinateur avec l'algorithme Minimax
  5. Implémenter et intégrer l'élagage $\alpha\beta$
  6. Implémenter plusieurs difficultés de jeu
     - Plusieurs profondeurs de recherche
     - Plusieurs fonctions d'évaluation

En pratique

- Rapport à rendre **avant le 23/04/2021**
  - Description du jeu programmé, et de ses règles
  - Description détaillée des IAs implémentées, explications des choix effectués
  - Bilan du projet
- Soutenances courant mai, après les examens, en fonction du programme des soutenances de projet tuteuré
  - Démonstration du jeu
  - Questions sur le code (*qui doit donc être propre et bien commenté...*)