

Recherche dans les arbres de jeux

Plan

□ Introduction

- Généralités
- Formalisme et représentation
- Recherche d'une stratégie gagnante

□ Algorithmes de recherche du meilleur coup

Généralités

- ❑ Les jeux ont toujours intéressé les informaticiens
- ❑ La pratique des jeux est universelle
 - Confrontation entre plusieurs personnes (compétitions)
 - La victoire dans les jeux avec stratégie est censée démontrer une certaine intelligence
- ❑ Les jeux sont facilement formalisables en informatique
 - ❑ Monde clos
 - ❑ Règles simples

Type de jeux

type information	déterministe	hasard
complète	échecs, go, tic-tac-toe	backgammon, petits chevaux
partielle		poker, bridge, scrabble

Jeux les plus courants

- ❑ Jeux déterministes à information complète (le jeu de l'adversaire est connu)
 - ❑ le tic-tac-toe (morpion)
 - ❑ le Nim ou de Marienbad
 - ❑ Le puissance 4
 - ❑ les dames
 - ❑ les échecs
 - ❑ Le défi des années 70 - 00
 - ❑ le go
 - Le prochain grand défi
- ❑ Jeux faisant intervenir le hasard
 - ❑ Backgammon (dés)
 - ❑ Le poker (info. incomplète)
 - ❑ Le bridge (info. incomplète)

Nous considérerons dans la suite de ce cours uniquement les jeux déterministes, à information complète, avec un adversaire

Type de recherche dans les arbres de jeu

❑ Analyse exhaustive

- calcul de l'ensemble des coups possibles
- possible seulement dans des jeux dont le nombre d'états est restreint (ex. tic-tac-toe)
- Recherche d'une stratégie gagnante si elle existe

❑ Recherche informée

- Dans le cas du jeu d'échecs, le nombre de positions et d'opérations explose
- Nécessité de guider la recherche par heuristique

❑ Recherche probabiliste

- Dans le cas du poker, des informations probabilistes doivent être prises en compte

❑ Nous considérerons dans la suite de ce cours uniquement les jeux déterministes à information complète

Principe

- ❑ Dans les jeux à information complète :
 - ❑ On se trouve dans une situation où deux adversaires s'affrontent
 - ❑ Contraintes :
 - Impossible de connaître le coup de l'adversaire
 - Temps d'action limité -> besoin d'approximation
 - ❑ On ne recherche en général pas une solution, mais un coup à jouer (le meilleur)
 - ❑ Principe du « look-ahead » :
 - ❑ On raisonne en développant à l'avance un certain nombre de coups et les différentes ripostes possibles de l'adversaire
 - ❑ On évalue les situations feuilles par une heuristique
 - ❑ On calcule par min-max le meilleur coup à jouer. La qualité du coup dépend de l'heuristique et de la profondeur du look-ahead (elle s'améliore si on augmente la profondeur).

=> Représentation par un graphe d'états

Recherche dans les arbres de jeu

Plan

□ Introduction

- Généralités
- Formalisme et représentation
- Recherche d'une stratégie gagnante

□ Algorithmes de recherche du meilleur coup

Formalisme

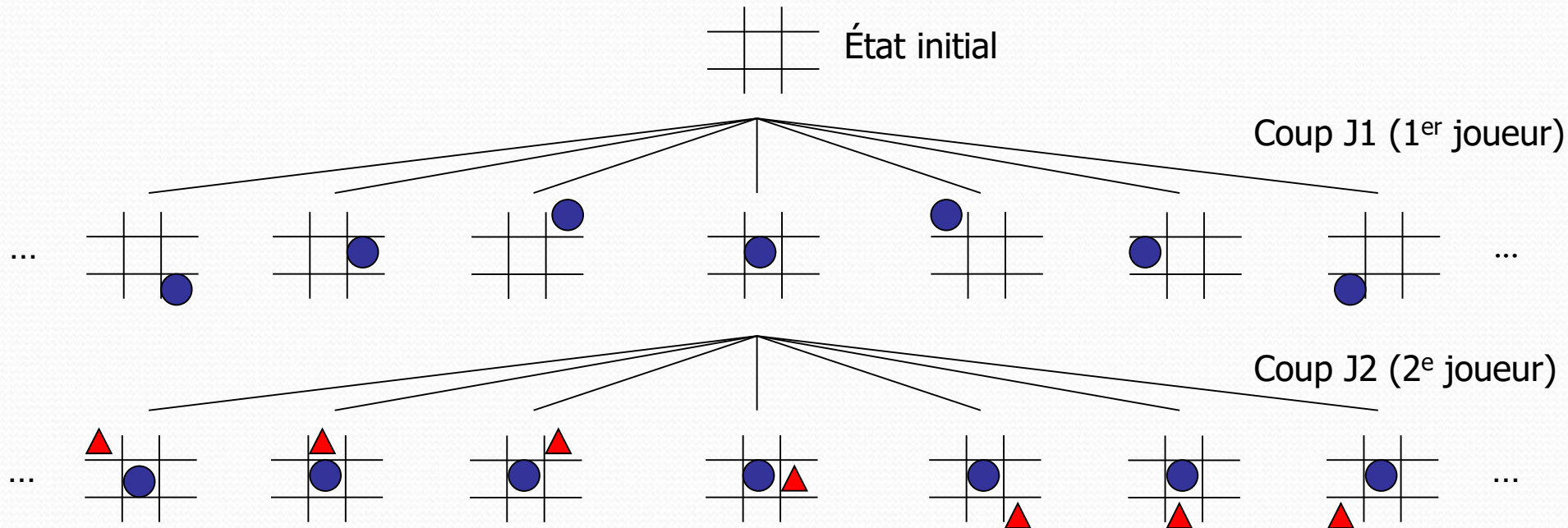
- ❑ Soit J_1 et J_2 deux joueurs, les règles du jeu permettent de définir un *arbre de jeu* :
 - La racine (profondeur 0) représente la position de départ
 - Les nœuds de profondeur paire : situation où J_1 doit jouer
 - Les nœuds de profondeur impaire : situation où J_2 doit jouer
 - Les arcs représentent les différents coups possibles
 - Les feuilles sont les positions gagnantes, perdantes ou bloquées

Lien avec les graphes d'états

- ❑ Le problème peut être représenté en terme d'opérateur de changement d'état :
 - La racine est l'état initial
 - Les opérateurs de changement d'états sont les coups légaux (alternance de J_1 et J_2)
 - Les états terminaux sont les situations gagnantes, perdantes ou bloquées

- ❑ Chaque chemin correspond à une partie

Exemple : tic-tac-toe



Attention aux symétries ...

Exemple : tic-tac-toe

État initial

Coup J1 (1^{er} joueur)

Coup J2 (2^e joueur)

...

...

...

...

Exercice « Grundy's game »

« On dispose d'une pile de 7 pièces : chaque joueur doit diviser une des piles en deux piles inégales; le perdant ne peut plus jouer »

- ❑ Formaliser le problème (états, états terminaux)
- ❑ Faire l'arbre ET-OU du point de vue J_1 puis du point de vue J_2

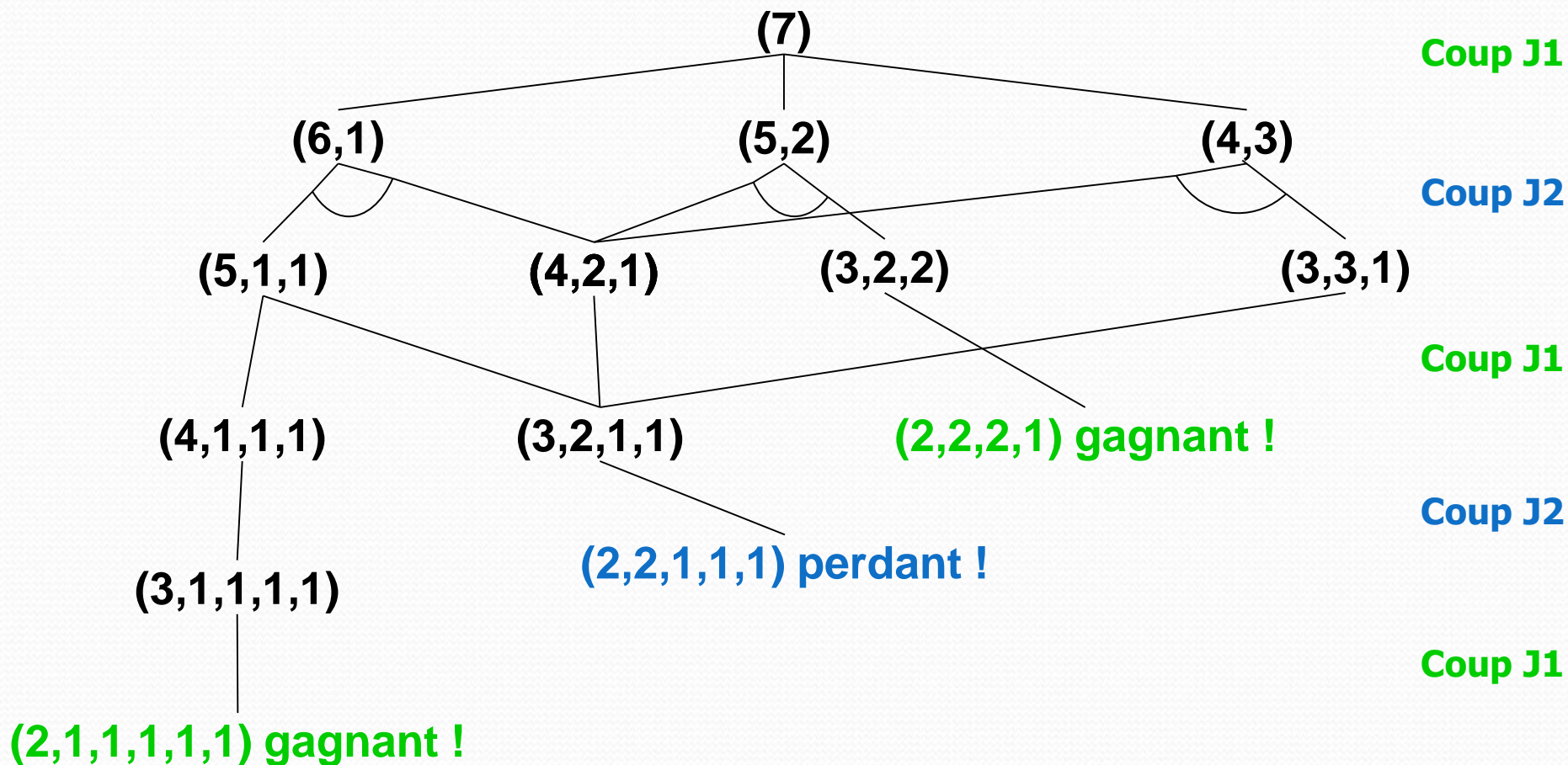
Exercice « Grundy's game »

« On dispose d'une pile de 7 pièces : chaque joueur doit diviser une des piles en deux piles inégales; le perdant ne peut plus jouer »

□ Arbre de recherche :

- État : liste de k chiffres ($k \leq 7$) $(n_1, n_2, n_3, \dots, n_k)$, avec $n_i \geq n_{i+1}$ le nombre de pièces dans chaque pile
- État terminal : lorsqu'il n'y a plus que des 1 ou des 2

« Grundy's game » : arbre de J1 (J1 commence + point de vue de J1)



Recherche dans les arbres de jeu

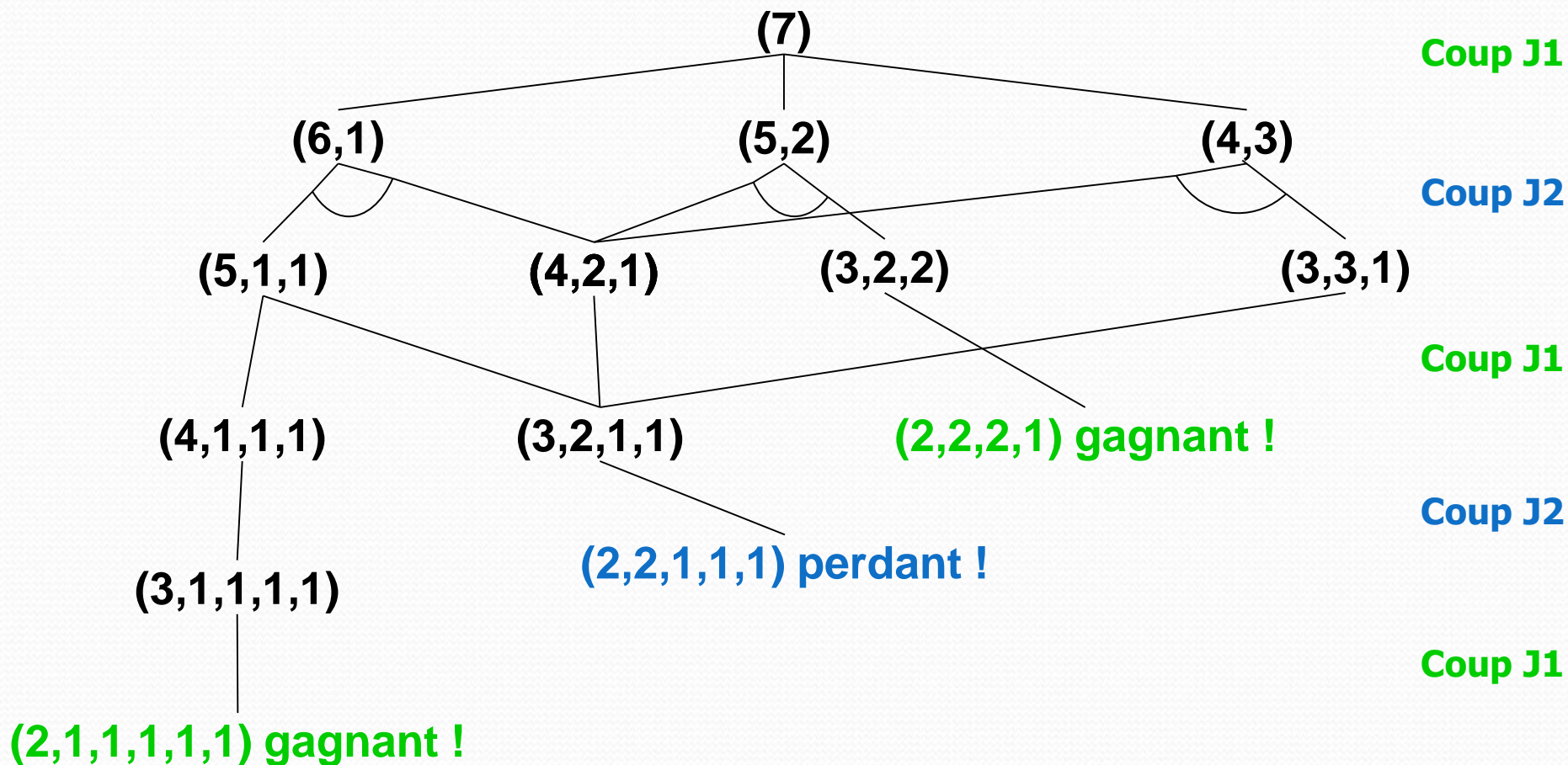
Plan

□ Introduction

- Généralités
- Formalisme et représentation
- Recherche d'une stratégie gagnante

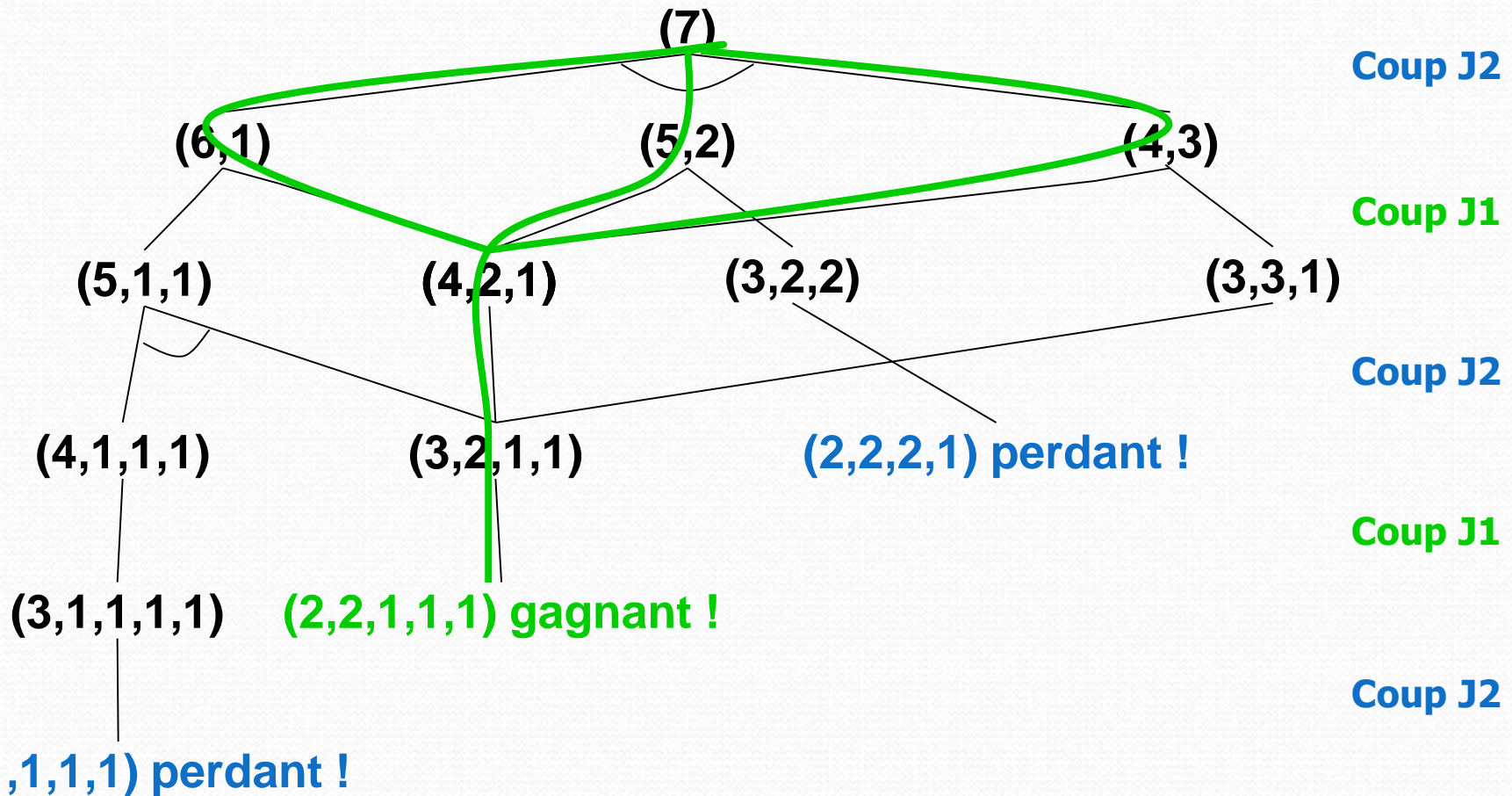
□ Algorithmes de recherche du meilleur coup

« Grundy's game » : arbre de J1 (J1 commence + point de vue de J1)



Existe il un arbre de jeu gagnant pour J1 s'il commence?

« Grundy's game » : arbre de J1 (J2 commence)



Existe il un arbre de jeu gagnant pour J1 si J2 commence?

Recherche d'une stratégie gagnante

- ❑ Idée : déterminer s'il existe une suite de coups qui mène à la victoire quelque soit le jeu de l'adversaire

- ❑ Un arbre de jeu est gagnant pour J_1 :
 - Si c'est une feuille victoire pour J_1
 - Ou si la racine est un nœud ET et que tous les fils sont gagnants
 - Ou si la racine est un nœud OU et un de ses fils est gagnant

=> Recherche d'une solution dans un arbre ET-OU

Recherche d'une stratégie gagnante (algorithme)

Fonction évaluer(racine R)? : existe (1 si existe, 0 sinon)

existe \leftarrow évaluer(R)

si FEUILLE(R) alors

si FEUILLE_GAGNANTE(R) alors existe \leftarrow 1

sinon existe \leftarrow 0 fsi

sinon si NŒUD_OU (R) alors existe \leftarrow max(évaluer(fils(R))

sinon si NŒUD_ET (R) alors existe \leftarrow min(évaluer(fils(R))

fsi

- On peut extraire l'arbre ET-OU gagnant en conservant les pères des noeuds
- On veut mettre un coût sur les coups et chercher la « meilleure » solution (la plus rapide) : voir AO*

Limite de la recherche de stratégie gagnante

❑ Stratégie gagnante très limitée :

- N'existe pas toujours pour les jeux complexes (heureusement !)
- Elle existe peut-être, mais l'arbre de recherche est trop gros pour le savoir (rappel taille = b^p , b =nb branchements, p =profondeur)

❑ Exemple :

- Dames : 10^{40} soit, si un nœud est traité en 1ns, $> 10^{22}$ siècles pour développer l'arbre : voir dames anglaises !
- Échecs : $b = 30$, $p = 100$ environ 10^{120} -> stratégie gagnante ?

=> On s'intéresse à la **recherche du meilleur coup** à jouer à un instant donné

Recherche dans les arbres de jeu

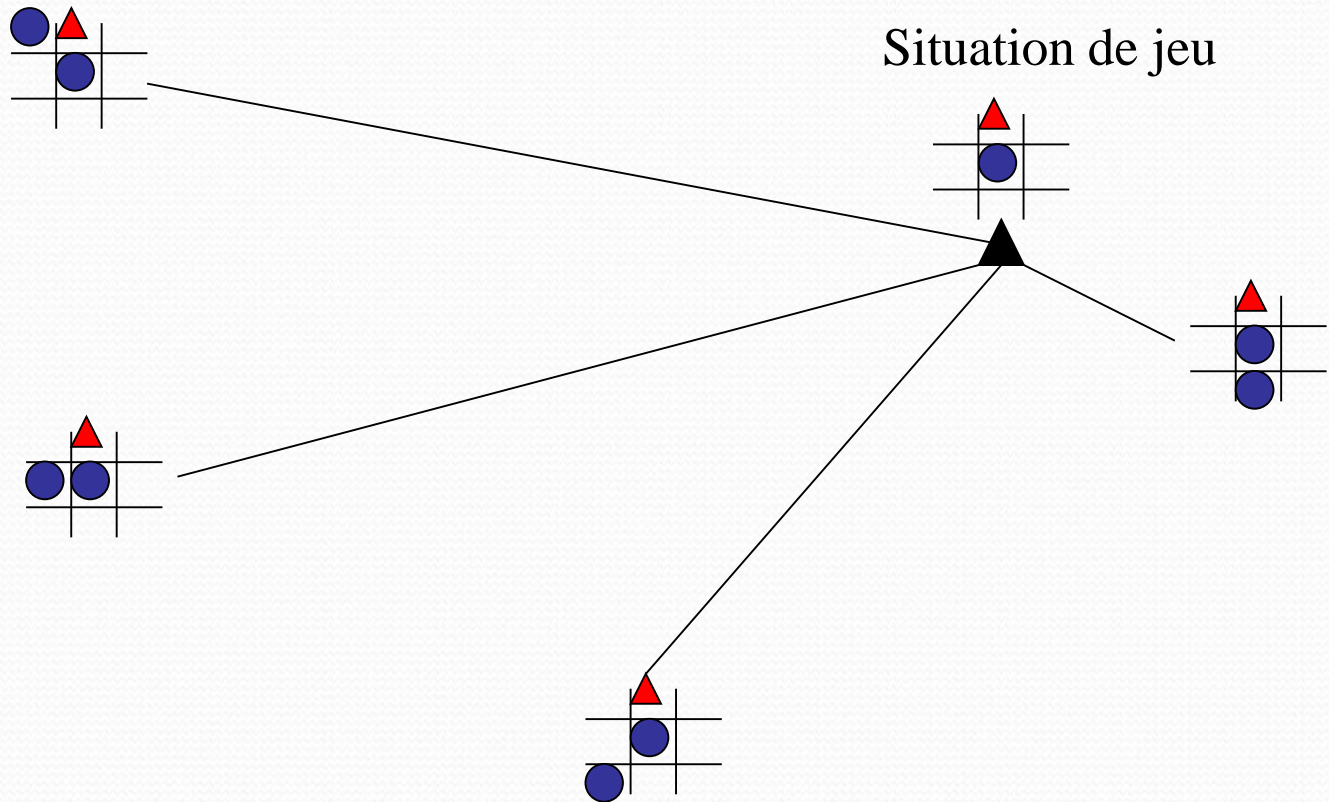
Plan

- Introduction
- Algorithmes de recherche du meilleur coup
 - Principe
 - Minimax
 - Alpha-beta
 - SSS*

Principe de la recherche du meilleur coup

- ❑ On ne recherche pas la « solution » (partie) optimale (voir AO*, voir reines anglaises)
- ❑ On s'intéresse à la détermination du meilleur coup à jouer à chaque pas
 - Seule une sous-partie de l'arbre de jeu est développée à une certaine profondeur pour déterminer le meilleur coup
 - Après riposte de l'adversaire il y a réévaluation de la stratégie
 - La qualité du choix dépend de la fonction d'évaluation des sous-arbres non développés

Exemple : tic-tac-toe



Quel est le meilleur coup à jouer pour ● ?

Fonctions d'évaluation

□ Choix heuristique du coup à jouer

- Développement à un seul niveau
 - Si un coup c est gagnant, le choisir
 - Si un coup c est perdant, le supprimer
 - Pour les coups restants, choisir le plus avantageux (ou le plus handicapant pour l'adversaire)
- **Technique du « looking ahead »**
 - On développe les nœuds jusqu'à une profondeur p_{\max} (fonction du temps de calcul par exemple) et on remonte l'évaluation jusqu'au nœud courant
 - Principe du Minimax

Recherche dans les arbres de jeu

Plan

- Introduction
- Algorithmes de recherche du meilleur coup
 - Principe
 - Minimax
 - Alpha-beta
 - SSS*

Minimax

□ Hypothèse de base

- La fonction d'évaluation est toujours du point de vue de J_1
- Le coup le plus intéressant est celui qui a la valeur maximale
- J_2 fait toujours les meilleurs choix pour lui (les plus contraignants pour J_1)

□ Évaluation

- La racine est la situation du jeu au moment où J_1 doit jouer
- On développe l'arbre jusqu'à une profondeur p_{\max} (sauf si..)
- Les feuilles sont évaluées et leur valeur est (rétro)propagée jusqu'à la racine
- Au nœud OU (choix de J_1) on associe le maximum des valeurs
- Au nœud ET (choix de J_2) on associe le minimum des valeurs (choix le plus contraignant pour J_1)

Algorithme du Minimax

$\alpha \leftarrow \text{maximin}(R)$

si FEUILLE(R) alors $\alpha \leftarrow h(R)$

sinon $\alpha \leftarrow \max(\text{Minimax}(\text{succ}_1(R)), \text{Minimax}(\text{succ}_2(R)), \dots, \text{Minimax}(\text{succ}_n(R)))$

fsi

$\beta \leftarrow \text{Minimax}(R)$

si FEUILLE(R) alors $\beta \leftarrow h(R)$

sinon $\beta \leftarrow \min(\text{maximin}(\text{succ}_1(R)), \text{maximin}(\text{succ}_2(R)), \dots, \text{maximin}(\text{succ}_n(R)))$

fsi

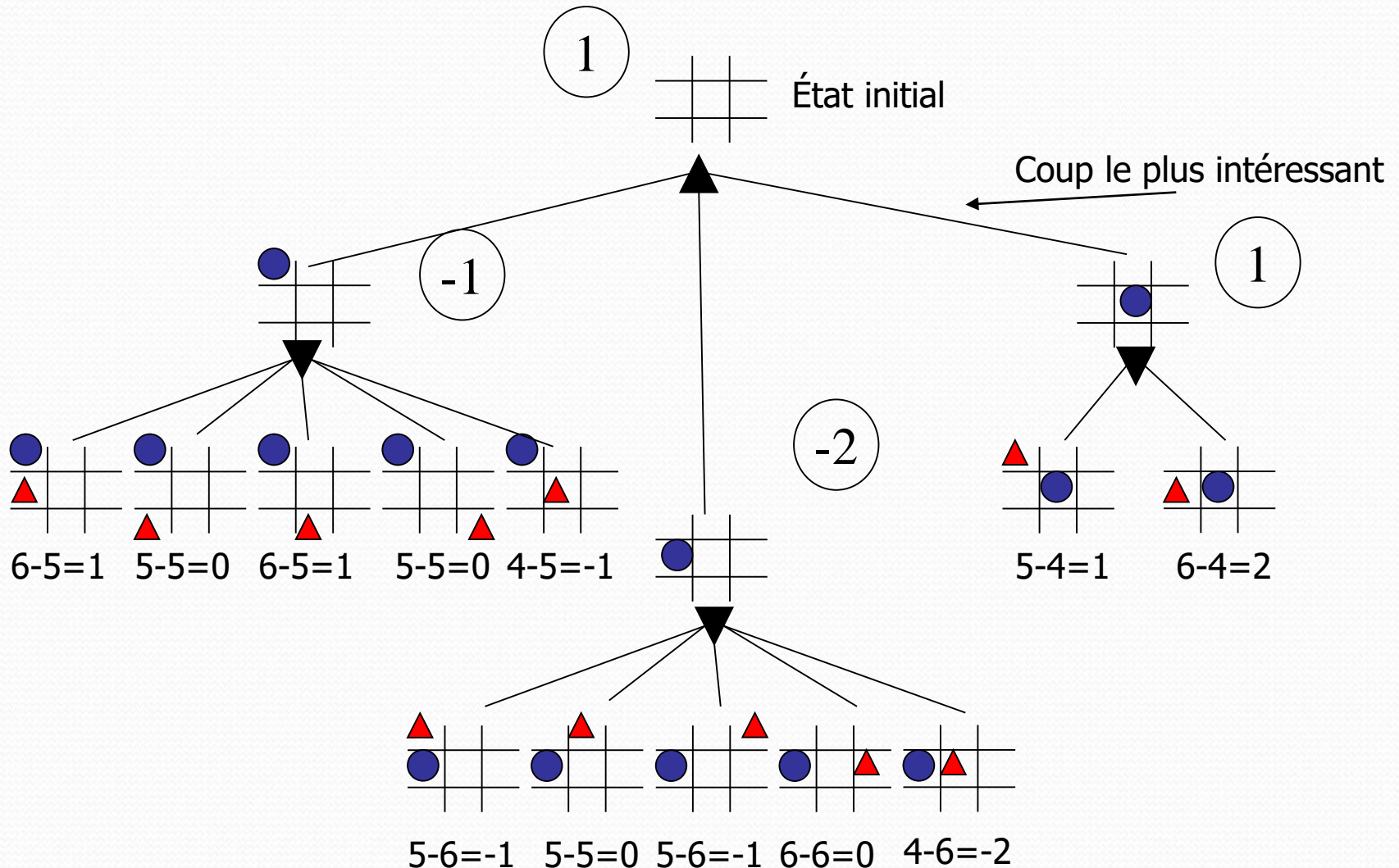
Remarques

- ❑ La notion de feuille est ici heuristique. Les feuilles ne sont pas obligatoirement toutes à la même profondeur (succès, échecs mais aussi attente d'une situation stable)
- ❑ L'heuristique ne sert pas ici à guider le développement de l'arbre.
 - L'arbre est développé complètement jusqu'aux feuilles
 - L'heuristique sert à valuer les feuilles

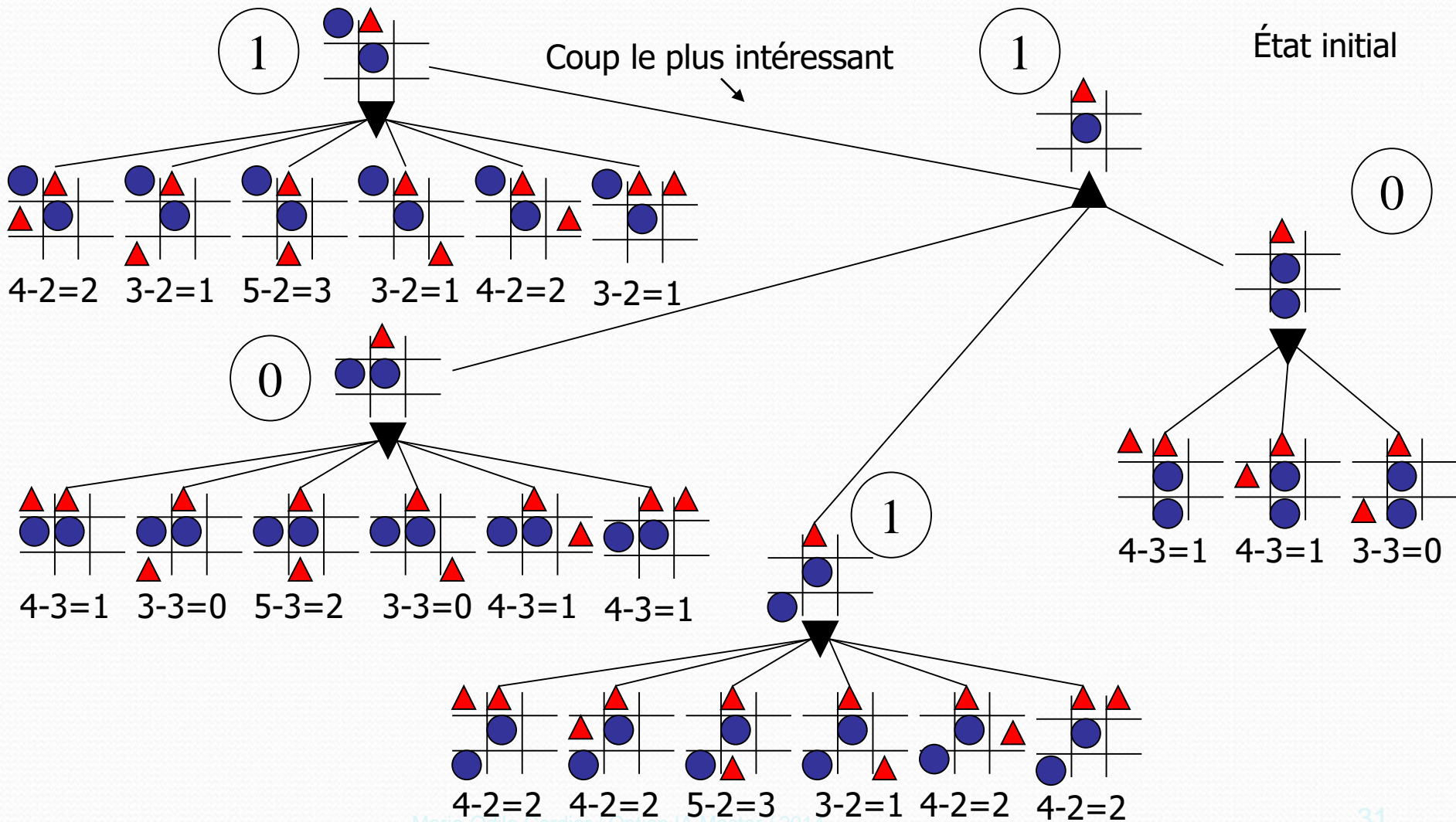
Exemple : tic-tac-toe

- ❑ Soit la fonction d'évaluation $h(F)$
 - $h(F) = \text{nb lignes} + \text{nb colonnes} + \text{diagonales ouvertes pour } J_1 - (\text{nb lignes} + \text{nb colonnes} + \text{diagonales ouvertes pour } J_2)$
- ❑ On considère une profondeur de développement de 2

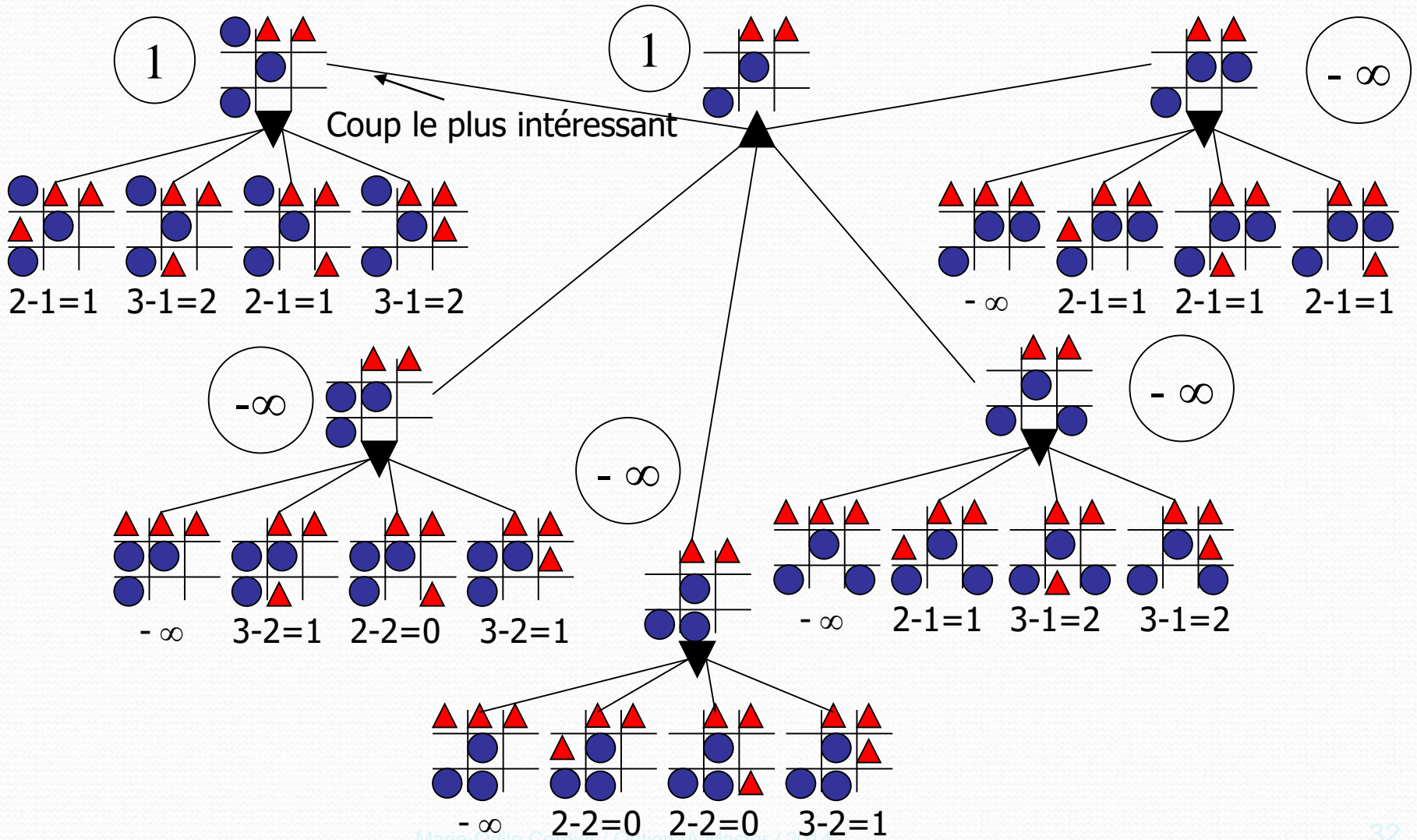
Exemple : tic-tac-toe



Exemple : tic-tac-toe



Exemple : tic-tac-toe



Propriétés de Minimax

- ❑ S'arrête toujours si l'arbre est fini
- ❑ Si b = nombre de coups possibles et p la profondeur moyenne des feuilles, minimax a une complexité en temps $O(b^p)$ et en espace $O(b \cdot p)$
- ❑ Problème d'horizon
 - Le meilleur coup à jouer à une profondeur p peut cacher un coup plus intéressant par la suite

Recherche dans les arbres de jeu

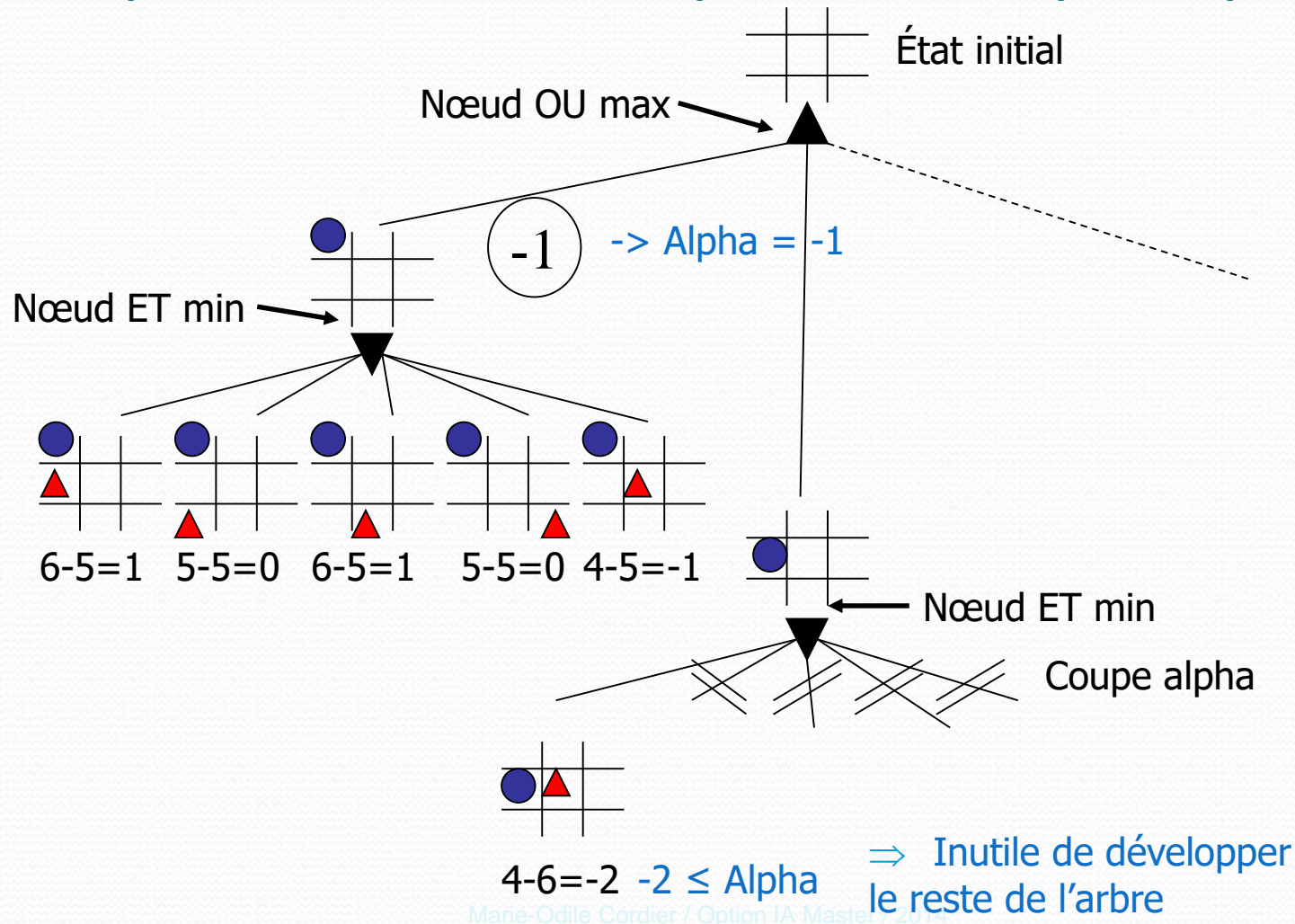
Plan

- Introduction
- Algorithmes de recherche du meilleur coup
 - Principe
 - Minimax
 - Alpha-beta
 - SSS*

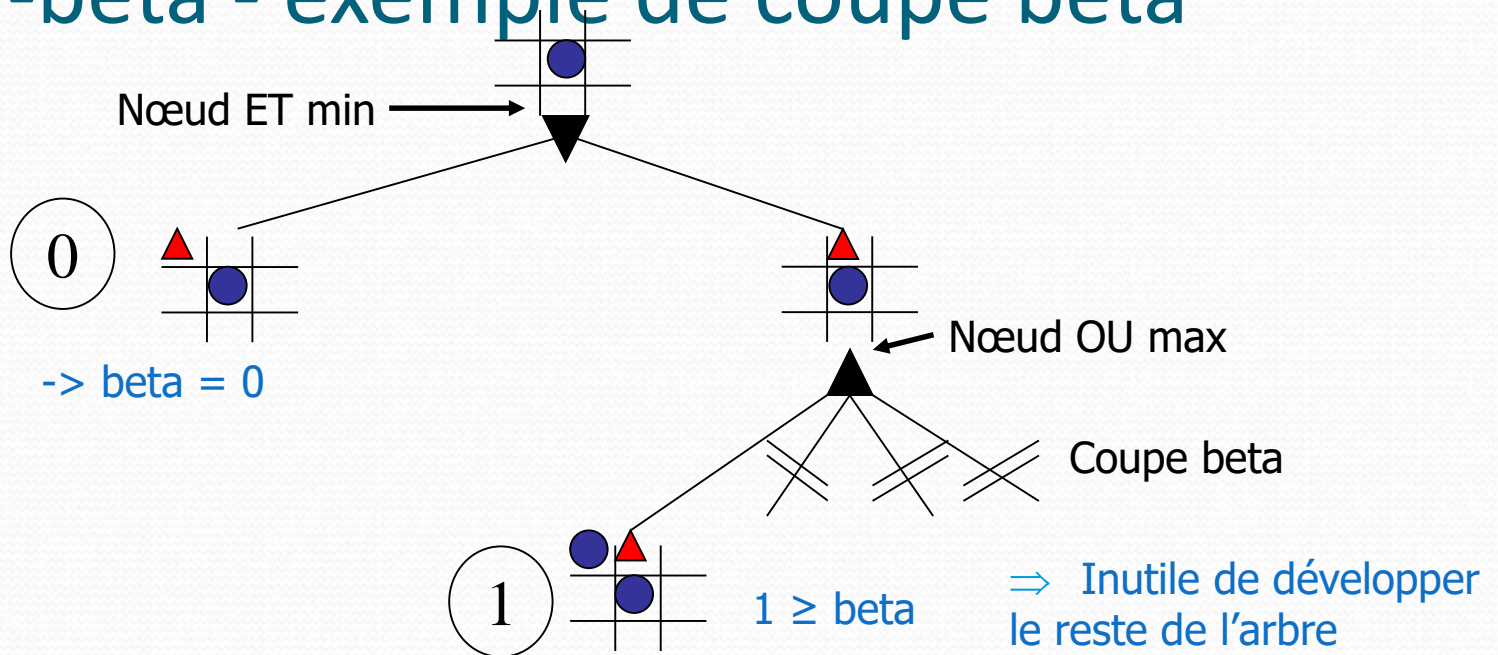
Alpha - Beta

- ❑ Le Minimax fait une énumération explicite de l'ensemble des coups possibles jusqu'à une certaine profondeur
- ❑ L'alpha-beta permet de ne développer que les noeuds intéressants à l'aide de coupes
- ❑ Deux types de coupes peuvent être envisagés
 - Coupe beta sur les nœuds OU
 - Coupe alpha sur les nœuds ET

Alpha-beta - exemple de coupe alpha



Alpha-beta - exemple de coupe beta



Algorithme de l'alpha-beta

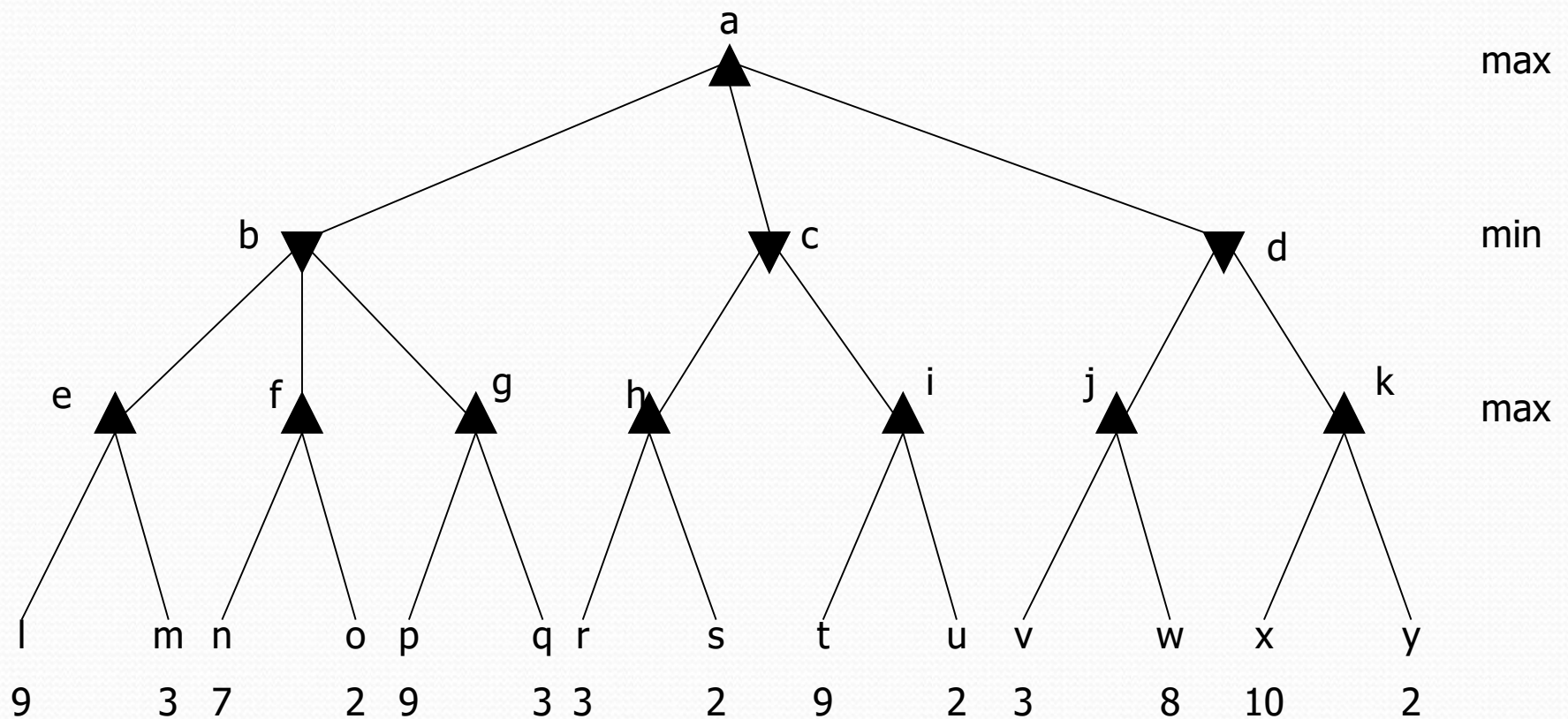
maximin (R,alpha,beta) // *R nœud max*

```
si FEUILLE(R) alors return h(R)
sinon
  eval =  $-\infty$ 
  pour tout successeur de R faire
    eval  $\leftarrow$  max(eval,
      minimax(succ(R), eval, beta))
  si eval > beta alors
    print « beta coupure »
    return eval
  fsi
fpour
return eval
fsi
```

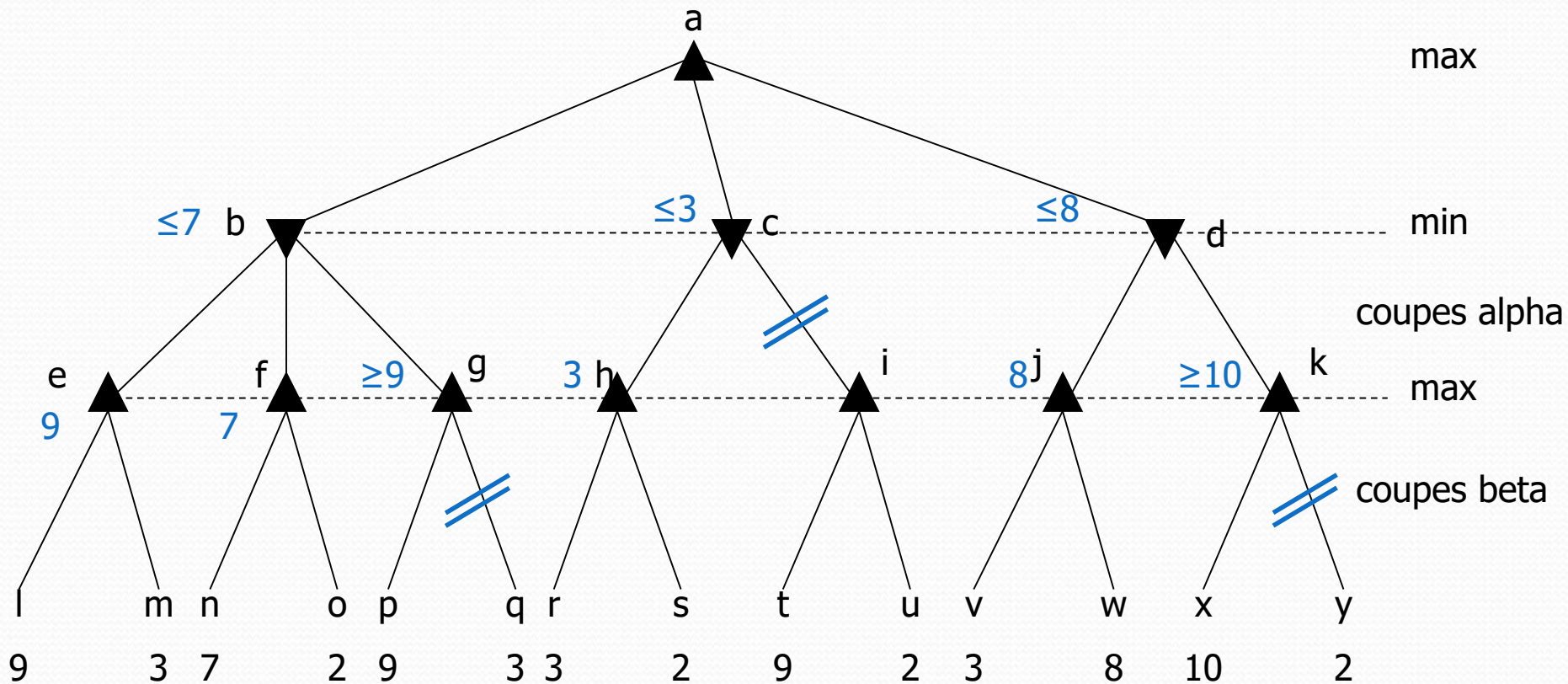
minimax (R,alpha,beta) // *R nœud min*

```
si FEUILLE(R) alors return h(R)
sinon
  eval =  $+\infty$ 
  pour tout successeur de R faire
    eval  $\leftarrow$  min(eval,
      maximin(succ(R), alpha,
        eval))
  si eval  $\leq$  alpha alors
    print« alpha coupure »
    return eval
  fsi
fpour
return eval
fsi
```

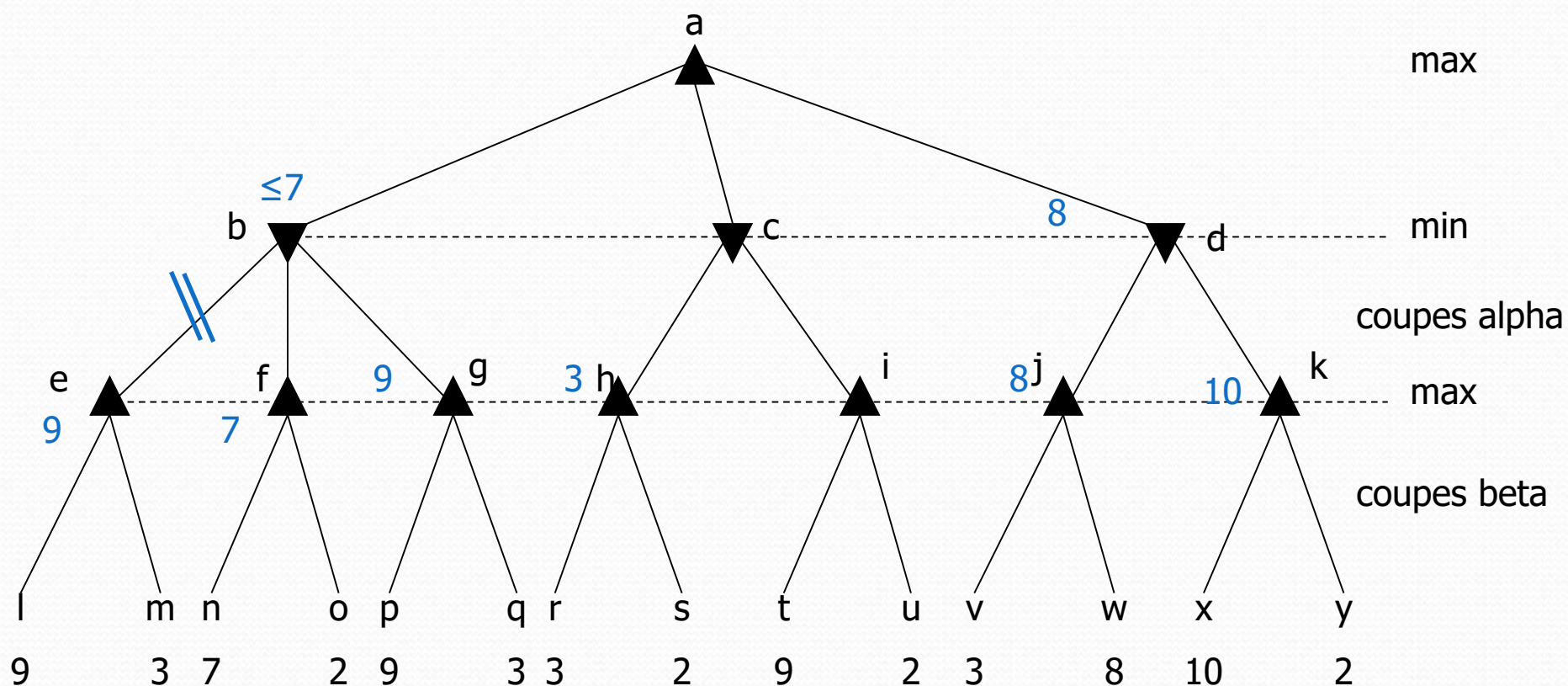

Alpha-beta – exercice



Alpha-beta – exercice



Alpha-beta – exercice (droite-gauche)



Remarques sur l'alpha-beta

- ❑ L'ordre dans lequel on visite les nœuds fils est important
 - Si on trouve rapidement une bonne valeur on élague plus de nœuds
 - Idée : Utiliser la fonction d'évaluation pour établir l'ordre de visite des nœuds fils

- ❑ Comment décider de la profondeur ?
 - Utiliser le principe de l'iterative deepening
 - On utilise le résultat obtenu pour ordonner les nœuds à l'itération suivante
 - Algorithme de type anytime (contrôle du temps passé pour chaque coup)